POLITECNICO DI TORINO

Master's Degree in CYBERSECURITY

Politecnico
di Torino

Master's Degree Thesis

Evaluating Backdoor Attacks Over Centralized and Distributed Medical Image Processing

Supervisors

Prof. Alessio SACCO

Prof. Guido MARCHETTO

Candidate

Christian CODURI

October 2025

Evaluating Backdoor Attacks Over Centralized and Distributed Medical Image Processing

Christian Coduri

Abstract

Machine learning, particularly deep learning models such as Convolutional Neural Networks, has shown great promise in medical imaging, supporting clinicians in diagnosis and treatment planning. However, clinical adoption is often limited by the scarcity of data and the risk of dataset bias, as models trained in a single institution may fail to generalize. In addition, data protection regulations, such as the GDPR, restrict the centralization of medical images across hospitals, limiting the development of robust models.

Federated Learning has emerged as a promising paradigm to address these challenges by enabling multiple institutions to collaboratively train a model without sharing raw data. In this approach, each institution updates the shared model using its own dataset and transmits only the parameters to a central server, thereby preserving the privacy of sensitive information.

In the future, it is reasonable to expect that only a few large, well-resourced hospitals, or institutions in isolated regions, will continue to develop their own centralized and independent models. In contrast, most hospitals and clinics are likely to increasingly adopt federated learning, enabling them to collaboratively train more powerful models while maintaining compliance with regulations.

Federated learning remains a variant of machine learning and therefore inherits many vulnerabilities while introducing new threats specific to its distributed nature. Among attacks common to both settings, data poisoning is concerning, with one of the stealthiest forms being backdoor attacks. In such attacks, adversaries embed hidden triggers into a subset of the training data, causing the model to behave normally on benign inputs but misclassify those containing the trigger, making detection challenging.

This study investigates the impact of backdoor attacks in both centralized and federated learning environments. Convolutional neural networks for brain tumor classification were developed using pre-trained architectures and transfer learning techniques. After assessing their performance under normal conditions, backdoor attacks were implemented on two selected models: VGG-16 and MobileNetV2 to evaluate their vulnerability and robustness.

The results show that, despite substantial architectural differences, the models exhibited similar behavior under backdoor attacks. With full fine-tuning, the models learned both medical features and trigger patterns, suggesting that implanting a trigger into the convolutional layers is relatively straightforward. Under this setting, high attack success rates (ASRs) were achieved: 99.89% for VGG-16 and 98.90% for MobileNetV2 when 10% of the training data was poisoned. In contrast, classifier-only fine-tuning effectively mitigated the impact of embedded triggers, reducing ASRs to 9.27% and 30.68%, respectively.

Following the single-agent experiments, the same models were deployed in a federated environment and exposed to attacks varying in the number of malicious clients and poisoning rates. Similar behavior to the centralized setting was observed under both IID and non-IID

data distributions: fully fine-tuned models achieved near 100% ASR (indicating effective trigger learning) while maintaining high clean accuracy, whereas classifier-only fine-tuned models again demonstrated strong resilience to trigger absorption.

Moreover, explainability techniques such as Grad-CAM have proven valuable during inference in both centralized and distributed settings, helping identify potential attacks by revealing when the model focuses on irrelevant image regions, possibly corresponding to a trigger, rather than medically meaningful areas. Finally, observations are presented on potential indicators of backdoor activity detectable across both environments.

Future work will build on this analysis to design robust defense mechanisms applicable to both centralized and distributed learning. Additional backdoor variants, including those with invisible triggers or adversarial noise, will be investigated and compared. Finally, attention will be given to developing defenses that remain effective under privacy-preserving techniques such as secure computation or differential privacy, ensuring both security and compliance in sensitive applications.

ACKNOWLEDGMENTS

My sincere thanks go to my supervisors, Professor Sacco and Professor Marchetto, for their guidance, from shaping the direction of this thesis to supporting decisions about my future.

I am also deeply grateful to my families: the one I was born into and the new one I am building, whose love and encouragement have always kept me moving forward.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

Healthcare is undergoing a profound digital transformation that is reshaping the traditional model of clinical history, examination, diagnosis, investigation, and treatment. This transformation is being driven by a range of emerging technologies, most notably artificial intelligence (AI), machine learning (ML), wearable sensors, mobile applications, and telehealth solutions.

Thanks to these technologies, rapid advancements have been achieved in tasks such as *screening* (e.g., early detection of cancers and other diseases), *prevention* (e.g., risk prediction), *therapy* (e.g., personalized treatment strategies), and *research and development* of new medicines, vaccines, and therapeutic approaches.

These opportunities, however, come with growing risks. Traditionally, clinicians have been valued for their long and rigorous training, which enables them to acquire and apply medical knowledge directly in patient care. Today, however, their role is evolving: rather than serving as sole decision-makers, clinicians are increasingly becoming interpreters of algorithmic recommendations [1].

Moreover, as healthcare systems increasingly rely on interconnected digital infrastructures and AI-driven technologies, they are becoming attractive targets for cybercriminals. This is not a hypothetical concern: in 2023 alone, EU countries reported 309 significant cybersecurity incidents in the healthcare sector, more than in any other critical sector, according to the European Commission [2].

This trend is expected to intensify with the emergence of increasingly sophisticated threats. While most current attacks target healthcare systems through data breaches and ransomware, the growing reliance on sensitive patient data further amplifies the risk of data-related compromises. At the same time, the integration of AI models introduces new vulnerabilities that adversaries can exploit through malicious manipulations, posing additional security and privacy challenges for modern healthcare.

1.1 Hospitals of the Future

One of the most concrete examples of this technological progress is the development of Computer-Aided Diagnosis (CAD) systems in radiology. Initially designed as visualization and annotation tools to assist clinicians, these systems have evolved into sophisticated platforms capable of automated image pre-processing, segmentation, and classification through the use of AI. Such AI-driven tools now enable faster and more accurate inter-

pretation of medical images, thereby enhancing clinical decision-making and improving patient outcomes.

The development of these systems relies heavily on deep learning methods, particularly *Convolutional Neural Networks* (CNNs), which are widely used for image-related tasks. However, training a CNN for medical image classification is often time-consuming and computationally demanding. Moreover, training deep learning models requires substantial resources, both financially and in terms of access to large, high-quality datasets.

A common strategy to alleviate these issues is to start from a *pretrained model*, typically trained on large, general-purpose datasets, and adapt it to the specific medical task using a technique known as *transfer learning*. This approach enables the model to be fine-tuned for the target domain while leveraging previously learned visual features.

In this context, two main training paradigms for CNNs are commonly employed: **centralized** and **distributed** (or federated) medical image processing. These approaches are expected to play a key role in shaping the hospitals of the future, giving rise to two distinct types of healthcare institutions:

- **Independent hospitals**, which adopt centralized learning. Here, medical data are stored locally and used to train CNNs within the institution. This approach gives hospitals direct control over their data but often comes with higher costs, limited dataset sizes, and potential population-specific biases.
- **Interconnected hospitals**, which collaborate through federated learning. Here, multiple institutions jointly train a shared global CNN model without exchanging sensitive patient data, thereby ensuring compliance with privacy regulations. Each institution sends locally computed model updates, reflecting its own patient population, to a central server for aggregation. This strategy enhances generalization, reduces bias, and preserves privacy.

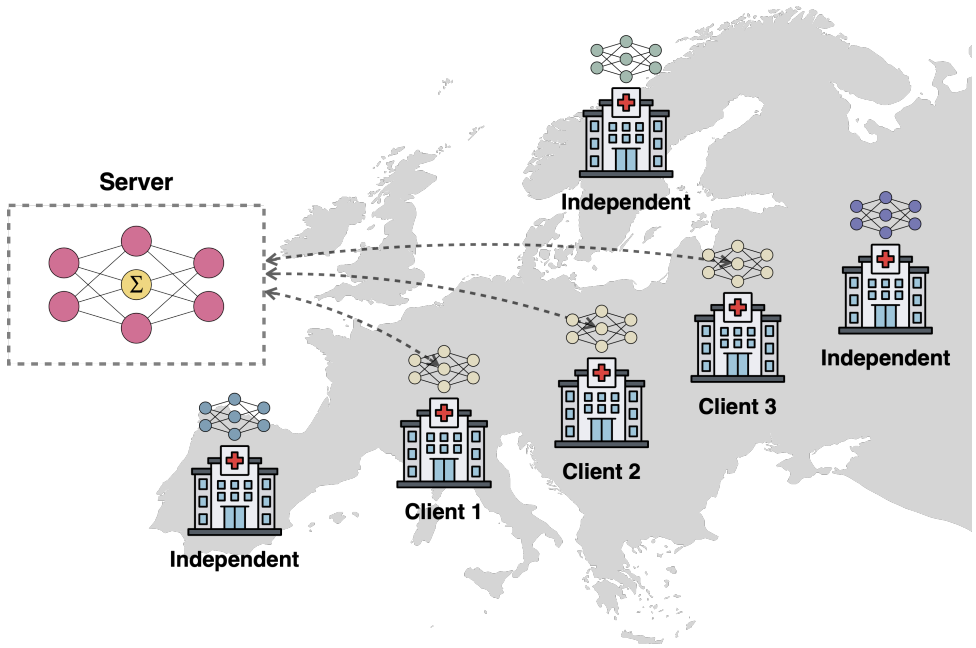


Figure 1.1: Illustration of potential future hospital configurations across Europe[1]

[1]This figure has been designed using resources from `https://flaticon.com`

1.2 Thesis Overview

CNNs used as classification systems can serve as powerful diagnostic tools; however, as additional software components, they also introduce vulnerabilities and new attack vectors that must be carefully managed. Among these, one of the most dangerous and stealthy threats, relevant to both centralized and federated CNN paradigms, are **backdoor attacks**. Their stealth lies in the fact that, once the backdoor is installed, the model continues to perform normally on clean data, maintaining high accuracy, while the malicious behavior is triggered only when inputs contain a specific pattern. In such cases, the model misclassifies the targeted samples while appearing reliable under standard conditions.

In clinical practice, the presence of backdoors and the resulting misclassification of medical images could have severe consequences, including missed diagnoses, delayed treatments, or inappropriate therapeutic decisions that directly *endanger patient safety*. The risks, however, extend beyond individual outcomes. In large-scale or pandemic scenarios, such as population-wide screening and monitoring during COVID-19, a successful backdoor attack could *compromise public health efforts*, distort epidemiological data, and obstruct timely interventions.

Objective

Several defense mechanisms against backdoor attacks have been proposed in the literature. However, most are specific to either centralized or distributed learning environments and therefore lack general applicability. Furthermore, only a few studies address scenarios involving pre-trained models, which are increasingly common in medical imaging.

This thesis investigates backdoor attacks on classification models for brain tumor detection in the two types of hospitals envisioned for the future: some operating centralized systems and others adopting federated learning. Specifically, it examines the behavior of two widely used pre-trained CNNs, VGG-16 and MobileNetV2, in both centralized and federated environments, evaluating their vulnerability to backdoor attacks through extensive experimentation.

By systematically comparing the two models under the two scenarios, this study aims to identify general principles to guide the design of robust and broadly applicable mitigation and detection techniques against backdoor attacks.

Research Questions

The contribution of this work is guided by the following research questions, which shaped both the experimental design and the subsequent analysis:

1. **Impact of model architecture and fine-tuning strategy:** How do different pre-trained CNN architectures perform in brain tumor classification under normal conditions, and how does the choice of training strategy (full fine-tuning vs. classifier-only fine-tuning) influence model accuracy and computational efficiency?

2. **Susceptibility to backdoor attacks in centralized learning:** How do the selected pre-trained models, VGG-16 and MobileNetV2, respond to backdoor attacks

in a centralized setting, and how do factors such as training strategy, learning rate, poisoning rate, and model architecture affect attack success and detectability?

3. **Comparative analysis of federated and centralized learning:** Can pre-trained models be effectively integrated into federated learning frameworks, and how does their performance under both normal and backdoor attack conditions compare to that in centralized training?

4. **Effect of training parameters on backdoor propagation:** How do variables such as training strategy, poisoning rate and the number of malicious clients influence the effectiveness of backdoor attacks and the robustness of federated models? Additionally, how does data distribution, IID versus Non-IID, impact model behavior in both scenarios?

5. **Explainable AI for backdoor detection and mitigation:** Can Explainable AI techniques, such as Grad-CAM, be employed to detect and mitigate backdoor attacks in both centralized and federated learning environments?

Structure

The thesis is organized into five chapters:

- **Chapter 1** provides an introduction to the research topic, outlining the motivation, objectives, and research questions that guide this work.

- **Chapter 2** presents the theoretical background necessary to understand the conducted experiments. It begins with an overview of medical image processing in radiology, followed by key concepts in machine learning and deep learning. The chapter concludes with detailed discussions on convolutional neural networks, federated learning, and backdoor attacks, which form the core foundation of this thesis.

- **Chapter 3** offers a concise review of the main categories of defense mechanisms against backdoor attacks, considering both centralized and distributed training environments.

- **Chapter 4** describes the experimental setup, including the dataset used and the data preprocessing pipelines. It presents the results of centralized learning experiments under normal conditions, which serve to identify the most suitable model for further analysis. The following sections report the outcomes of backdoor attack experiments in both centralized and federated settings, each accompanied by an evaluation of the models' performance and robustness under varying conditions and some considerations about possible mitigation and techniques.

- Finally, **Chapter 5** summarizes the main findings and outlines potential directions for future research.

Chapter 2

Background

2.1 Medical Image Processing and AI in Radiology

Medical imaging plays a vital role in precision medicine by enabling accurate diagnosis and personalized treatment. In the field of radiology, the primary imaging modalities commonly used in clinical practice include X-rays, computed tomography (CT), magnetic resonance imaging (MRI), positron emission tomography (PET), and ultrasound.

Interpreting these images remains a complex task due to challenges such as image noise, irrelevant visual artifacts, and the intricate nature of disease presentations. These factors can contribute to diagnostic errors, including both false negatives and false positives. As reported in [3], human error in medical image interpretation can result in misdiagnosis rates ranging from 10% to 30%.

To support clinicians in reducing diagnostic errors, computer-aided diagnosis (CAD) systems have been developed. Modern CAD systems can be classified into two main categories:

- **CADe** (Computer-Aided Detection): Designed to locate and highlight potentially suspicious regions within medical images. It serves as a supplementary tool for radiologists, helping to reduce false positive detections, that is the likelihood of missed abnormalities.
- **CADx** (Computer-Aided Diagnosis): Aims to further analyze the detected regions by characterizing them and estimating the probability of disease, thereby supporting more accurate diagnostic decisions.

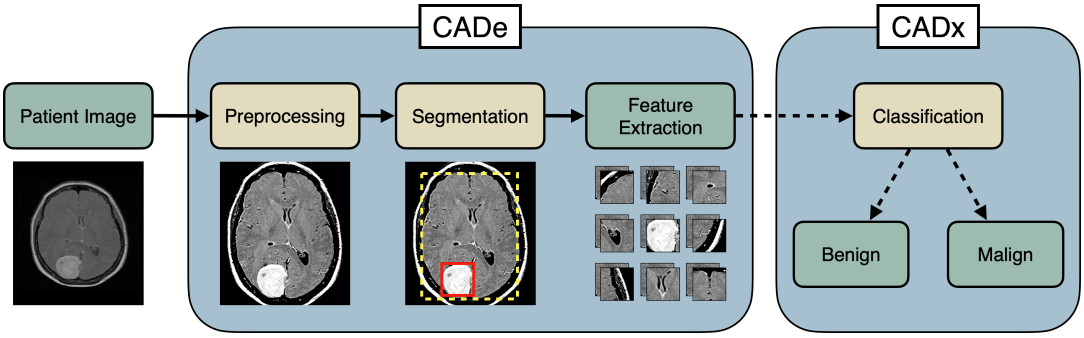


Figure 2.1: Computer-Aided Detection System

2.1.1 Enhancing Diagnostic Capabilities with AI

Although research interest in traditional CAD systems is decreasing in recent years, the number of studies applying artificial intelligence (AI) to medical image analysis has increased significantly. AI extends the capabilities of conventional CADx systems by transforming medical images into rich, mineable data [4].

Its applications have expanded across various imaging modalities, particularly in radiology, where AI has demonstrated substantial potential in detecting and classifying abnormalities on plain radiographs, CT, and MRI scans. These advancements contribute to more accurate diagnoses and better-informed treatment decisions.

Radiology is particularly well-suited for the integration of AI tools into clinical workflows, thanks to its mature digital infrastructure and standardized protocols for image acquisition and storage. Moreover, AI excels at well-defined tasks such as anatomical segmentation, quantification of structures or abnormalities, identification of suspicious findings, and the detection, localization, and classification of various medical conditions [5].

2.1.2 Today a Co-Pilot, But Tomorrow?

At present, most AI algorithms in radiology are used as tools to enhance pre-interpretive processes, primarily supporting radiologists rather than replacing their expertise. In fact, according to [5], the majority of radiologists view AI as a *co-pilot* a collaborative second reader within the human-AI workflow.

However, some studies have shown that AI systems, particularly for chest radiograph interpretation, can achieve performance comparable to board-certified radiologists when used by non-radiology clinicians. This suggests that, in the future, it may be possible to achieve fully autonomous ML-based image analysis.

In the following sections, we introduce the fundamentals of deep learning and convolutional neural networks (CNNs), which are the most widely used machine learning models for image classification and the core technology employed in this thesis. CNNs have been shown not only to enhance radiologists' performance but, in some cases, to outperform multiple radiologists working together. For example, a 2021 study trained three CNN models, ResNet-18, ResNet-50, and DenseNet-201, using a dataset of 3,000 chest X-ray images, half from COVID-19 patients and half from healthy individuals [6]. When evaluated on a separate test set of 250 images, these models outperformed two radiologists across multiple metrics, including accuracy, sensitivity, specificity, precision, and F1-score. Similarly, a 2019 systematic review of over 30,000 studies concluded that deep learning models generally match or slightly outperform healthcare professionals across a wide range of medical classification tasks [7].

Beyond image analysis, AI's impact extends into diagnostic reasoning. Recent work by McDuff et al. [8] demonstrated that a large language model (LLM) optimized for clinical decision-making can outperform clinicians, even those assisted by search tools or the same LLM. Similarly, Microsoft has outlined the idea of Medical Superintelligence, describing AI systems capable of reasoning through complex clinical cases in ways resembling but, again, outperforming expert physicians [9].

These findings underscore the growing potential of AI to complement and, in some cases, surpass human expertise in medical diagnostics. If such advances become more

significant in the future, clinicians may increasingly rely on AI-generated analyses without question, highlighting the critical relevance and importance of this thesis.

2.2 Machine Learning

Machine Learning (ML) is a branch of Artificial Intelligence (AI) that enables machines to learn from data by recognizing patterns and making decisions with minimal human intervention. A key strength of ML models lies in their ability to adapt autonomously, improve through experience, and generalize effectively when exposed to new data.

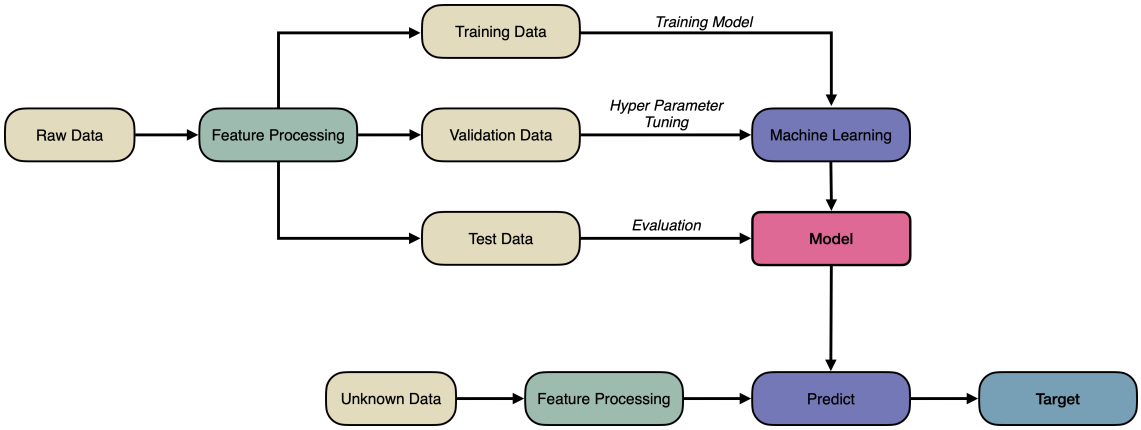


Figure 2.2: Machine Learning Process

In Figure 2.2, the general machine learning workflow is illustrated. After collecting and creating a dataset of raw data, these undergoes feature processing to prepare data and extract meaningful characteristics that will be used for the learning process. The processed data are then split into training, validation, and test sets.

- **Training set** is used to train the machine learning model by allowing the algorithm to learn patterns and relationships.
- **Validation set** is used during training for hyperparameter tuning, which involves adjusting non-learnable configuration settings to improve model generalization and prevent overfitting.

Once the model has been trained and tuned, it is evaluated using the **test set**, a separate subset not seen during training or validation. This final evaluation provides an unbiased estimate of the model's performance in real-world scenarios.

In the deployment phase, the trained model is used to make predictions on unknown data, new instances that also undergo the same feature processing pipeline to ensure consistency. The model generates predictions, which are then compared to actual target values if available, or used for downstream decision-making.

This workflow follows the **offline learning paradigm**, where the model is trained once on a fixed dataset and then deployed without further updates. If new data becomes available and an update is needed, the model must be retrained, either from scratch or by fine-tuning an existing model, using the new data, often combined with the original training set to preserve performance and avoid forgetting earlier knowledge.

The opposite approach is known as **online learning**, a paradigm in which the model is continuously updated as new data arrives. This allows the system to adapt to evolving patterns in real time but requires careful management to maintain stability and prevent performance degradation.

Unfortunately, traditional ML approaches typically rely on structured data and rigid, predefined workflows, where missing or poorly engineered features can significantly degrade performance. These limitations become especially apparent when dealing with complex, high-dimensional data such as medical images.

To address these challenges, more advanced approaches like deep learning have gained prominence, offering greater flexibility and performance in tasks involving unstructured data [10].

2.3 Deep Learning

Deep learning is a subset of machine learning that employs artificial neural networks with multiple layers to automatically learn hierarchical representations from data. It has achieved remarkable success across various domains, including computer vision and natural language processing.[1]

Common deep learning architectures include Convolutional Neural Networks, Recurrent Neural Networks (RNNs), and Generative Adversarial Networks (GANs). This section provides an overview of the fundamental concepts underlying CNNs, which constitute the core focus of this thesis.

2.3.1 Perceptron

Inspired by biological neural networks, a perceptron is a machine learning algorithm that takes input features and their corresponding targets, attempting to find a line, plane, or hyperplane that separates classes in two, three, or higher-dimensional space, respectively.

A perceptron consists of nodes that communicate with other nodes through connections weighted according to their contribution to achieving a desired outcome.

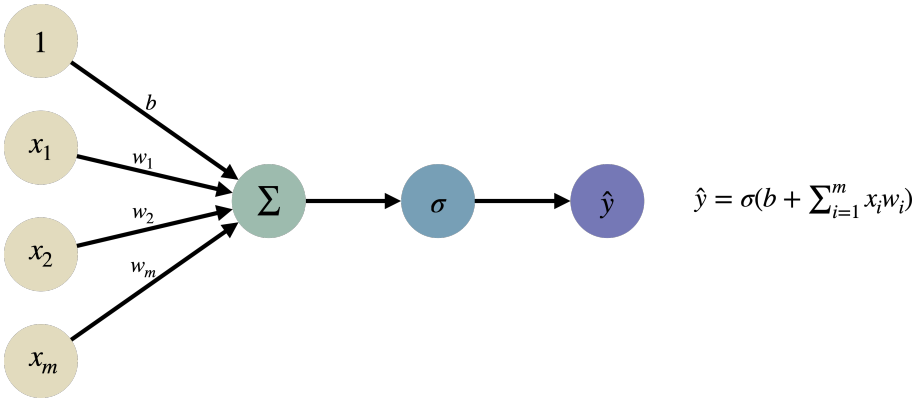


Figure 2.3: Perceptron: the Single Neuron

Given an input vector $X = (x_1, x_2, \ldots, x_m)$ representing a single instance with m features, and a corresponding weight vector $W = (w_1, w_2, \ldots, w_m)$, the output $\hat{y}$ is computed

[1] Introduction to Deep Learning. MIT 6.S191: `https://youtu.be/alfdI7S6wCY`

as $\hat{y} = \sigma(b + X^T W)$, where b is a bias term and σ is an activation function.

In other terms, a perceptron performs the following three main operations:

1. **Dot Product:** Each input is multiplied by its corresponding weight, and the results are summed to compute the dot product between vectors X and W.
2. **Add Bias:** A bias term b is added to the dot product to allow the activation function more flexibility by shifting the result. This helps the perceptron make adjustments independent of the input.
3. **Apply Non-Linearity:** The final result is passed through a non-linear activation function σ, producing the neuron's output. This step introduces non-linearity into the model's output, which is crucial for enabling the model to generalize beyond linear patterns and learning complex patterns in data.

2.3.2 Artificial Neural Networks

When multiple perceptrons are connected, the model becomes a multilayer perceptron or artificial neural network (ANN). ANNs typically consist of an input layer, one or more hidden layers, and an output layer. Simple ANNs may have up to a few hidden layers, while deep neural networks can contain dozens or even hundreds. In most cases, information flows in one direction, from input to output, forming a **feedforward neural network**.

In a neural network, the number of layers on the X-axis reflects the *depth* of the model; typically, deeper networks are capable of learning more abstract representations of the input. Each layer in an ANN consists of multiple nodes, which determine the *dimensionality* of its output. Hidden layers encode intermediate features, while the output layer's dimensionality depends on the task: for example, one node per pixel for image generation, or one node per class for classification.

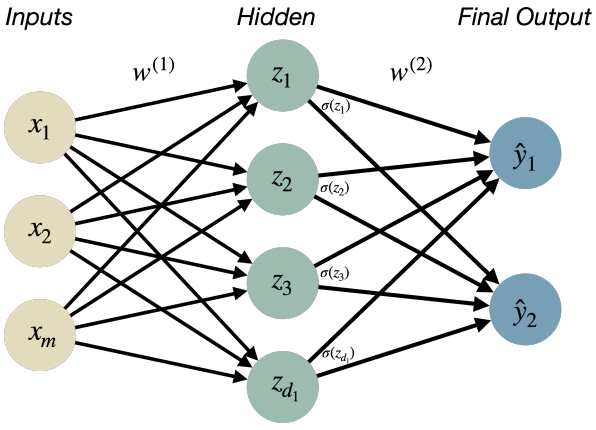


Figure 2.4: Artificial Neural Networks: Multi-layer Perceptron

We now consider the computation across two layers of weights. Let z_i represent the linear combination of inputs and weights before applying the non-linearity. For the first hidden layer, this can be written as:

$$z_i = w_{0,i}^{(1)} + \sum_{j=1}^{m} x_j w_{j,i}^{(1)}$$

The corresponding output $\hat{y}_i$, after passing through the hidden layer and applying the activation function σ, is computed as:

$$\hat{y}_i = \sigma\left(w_{0,i}^{(2)} + \sum_{j=1}^{d_1} \sigma(z_j) w_{j,i}^{(2)}\right)$$

Here, $w_{0,i}^{(1)}$ and $w_{0,i}^{(2)}$ are bias terms for the first and second layers, respectively, m is the number of input features, and d_1 is the number of hidden units in the first layer [11].

2.3.2.1 Activation Functions

In the output layer, the number of nodes and the choice of activation function depend on the task: binary classification typically uses a single node with a *sigmoid* activation, multiclass classification uses one node per class often with a *softmax* activation, and regression tasks commonly use a single node with a *linear* activation.

Activation functions are applied at each node throughout the network, not just in the output layer, to introduce non-linearity. Without them, no matter how many layers the network has, it would behave like a linear model and fail to capture the complex, non-linear patterns present in most real-world data [12].

The most commonly used activation functions are shown in Table 2.1.

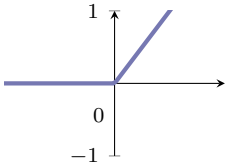
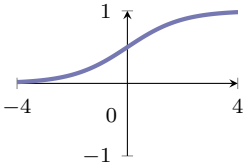
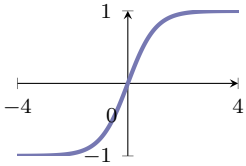
ReLU	**Sigmoid**	**Tanh**
$\sigma(z) = \max(0, z)$ 	$\sigma(z) = \frac{1}{1+e^{-z}}$ 	$\sigma(z) = \frac{e^{z}-e^{-z}}{e^{z}+e^{-z}}$ 

Table 2.1: Activation Functions

2.3.2.2 Loss

The loss measures how far the network's predictions deviate from the true values, quantifying the cost of incorrect predictions. Depending on the task, different loss functions are used, the three simplest loss functions are:

- **Squared Error Loss (MSE)**:
  Commonly used for regression, the squared error for a single prediction is $L = (\hat{y} - y)^2$. Over a dataset of n samples, the Mean Squared Error (MSE) is the average squared difference:
  $$\mathcal{L}_{\text{MSE}} = \frac{1}{n} \sum_{i=1}^{n} (\hat{y}_i - y_i)^2$$
- **Absolute Error Loss (MAE)**:
  Commonly used for regression, the absolute error for a single prediction is $L = |\hat{y} - y|$.

The Mean Absolute Error (MAE) over n samples is the average absolute difference:

$$\mathcal{L}_{\text{MAE}} = \frac{1}{n}\sum_{i=1}^{n}|\hat{y}_i - y_i|$$

- **Cross-Entropy**:
  Cross-entropy is used for classification problems to compare the predicted probabilities $\hat{y}$ with the true labels y. It penalizes incorrect predictions by measuring the difference between the predicted and true probability distributions.

  For a single data point in binary classification, where the number of classes is $M = 2$, the loss is:

  $$\mathcal{L}_{\text{BCE}} = -\left[y\log(\hat{y}) + (1-y)\log(1-\hat{y})\right]$$

  where $y \in \{0, 1\}$ is the true binary label.

  For a single data point in multiclass classification, where $M > 2$, the true label y is typically one-hot encoded[2], and the model outputs a probability distribution $\hat{y}$ over the M classes. The categorical cross-entropy loss is:

  $$\mathcal{L}_{\text{CCE}} = -\sum_{k=1}^{M} y_k \log(\hat{y}_k)$$

  which penalizes the model more when it assigns low probability to the correct class.

  When applied to an entire dataset of n samples, the overall loss is typically computed by averaging the per-sample loss values, as is done for MSE and MAE:

  $$\mathcal{L}_{\text{BCE}} = -\frac{1}{n}\sum_{i=1}^{n}\left[y_i\log(\hat{y}_i) + (1-y_i)\log(1-\hat{y}_i)\right]$$

  $$\mathcal{L}_{\text{CCE}} = -\frac{1}{n}\sum_{i=1}^{n}\sum_{k=1}^{M} y_k^{(i)}\log(\hat{y}_k^{(i)})$$

Each of these loss functions provides a numerical signal during training, guiding the optimizer to adjust model parameters in a direction that improves predictive accuracy (see next section).

2.3.3 Training ANNs: Gradient and Backpropagation

The goal of training a neural network is to find the set of weights W^* that minimize the chosen loss function:

$$\boxed{W^* = \arg\min_W \frac{1}{n}\sum_{i=1}^{n} L(f(x^{(i)}; W), y^{(i)}),} \tag{2.1}$$

The Equation 2.1 represents the process of minimizing the loss function across the entire dataset. Each component can be interpreted as follows:

[2] **One-hot encoding** is a technique used to represent categorical class labels as binary vectors, where only the position corresponding to the correct class is set to 1 and all others are 0.

- $\frac{1}{n}\sum_{i=1}^{n}$: This is an average over all n training samples. We want the average loss across the whole dataset to be as small as possible.
- $f(x^{(i)}; W)$: The model's prediction (or output) for the i-th input using parameters W, equivalent to what we previously denoted as $\hat{y}_i$.
- $y^{(i)}$ The true label or target output for the i-th training example.
- $L(f(x^{(i)}; W), y^{(i)})$: The loss function, which measures the difference between the model's prediction and the true label.

2.3.3.1 Gradient Descent

The loss is a scalar value. If the model has two weights, we can visualize the loss landscape as a 3D surface, where each point corresponds to a particular configuration of weights, and the height represents the loss value ($L = J(W)$). The goal is to find the values of w_0 and w_1 that minimize the loss.

Starting from a random point in the weight space, the gradient, a local measure of how the loss changes, is computed to determine which direction to "move in". By taking small steps in the opposite direction of the gradient, the loss is iteratively reduced, ideally converging to a minimum. Figure 2.5 illustrates an example of a 3D loss landscape (left), where a path highlights the trajectory of gradient descent. On the right, a 2D view traces the optimization path from start to finish, showing that the algorithm may not always reach the global minimum.

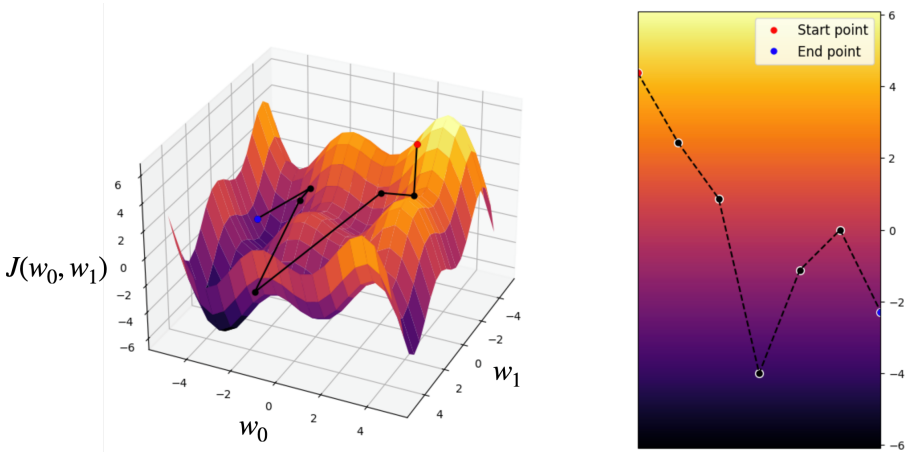


Figure 2.5: Gradient Descent 3D Plot Example

This optimization process, illustrated in the figure above, can be summarized in the following algorithm:

1. **Initialize weights randomly:** Initialize the weights $W^{(0)}$ at random, where $W^{(0)}$ represents the initial set of weights w_0 and w_1.
2. **Loop until convergence:** Iterate until the convergence condition is met, i.e., until the change in the loss function is smaller than a predefined threshold or a maximum number of iterations is reached:

- **Compute the gradient:** At each iteration t, compute the gradient of the loss function J with respect to the weights $W^{(t)}$. The gradient is given by:

$$\nabla_W J(W^{(t)}) = \frac{\partial J(W)}{\partial W^{(t)}}$$

- **Update the weights:** Update the weights $W^{(t)}$ using the computed gradient and a learning rate η. The update rule is:

$$W^{(t+1)} = W^{(t)} - \eta \nabla_W J(W^{(t)})$$

3. **Return weights:** Once the algorithm converges, return the final weights W^*:

$$W^* = W^{(T)}$$

where T is the total number of iterations (or the point of convergence).

2.3.3.2 Back-Propagation: Computing the Gradient:

The gradient indicates the direction in which the loss function increases or decreases and that computation is essential for updating the weights during training. The process of efficiently calculating the gradient in a neural network is known as *backpropagation*.

1. Compute the loss $J(W)$, representing the difference between the predicted and actual outputs.
2. Propagate the loss value backward through the network to each neuron.
3. Compute the gradient of the error with respect to each weight at every neuron.

Consider a simple feedforward neural network with a scalar input x, two weights w_1 and w_2, and a scalar output $\hat{y}$, as illustrated in Figure 2.6. The objective is to compute the gradient of the loss $J(W)$ with respect to each weight parameter w_1 and w_2 in order to update the weights.

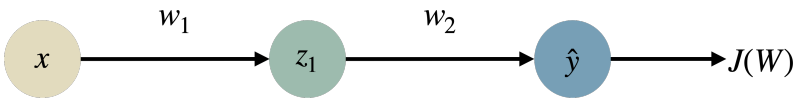


Figure 2.6: Back-propagation

To compute the gradients, we apply the **chain rule**. Specifically, the derivative of the loss with respect to each weight is decomposed into multiple partial derivatives. Since each weight depends only on the outputs of the previous layer, the gradients are computed starting from the output layer and propagated backward.

- For w_2, the gradient is: $\frac{\partial J(W)}{\partial w_2} = \frac{\partial J(W)}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial w_2}$
- For w_1, the gradient is: $\frac{\partial J(W)}{\partial w_1} = \frac{\partial J(W)}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial w_1} \cdot = \frac{\partial J(W)}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial z_1} \cdot \frac{\partial z_1}{\partial w_1}$

This procedure is applied to every weight in the network, allowing the error to be propagated backward using the computed gradients from subsequent layers. In doing so,

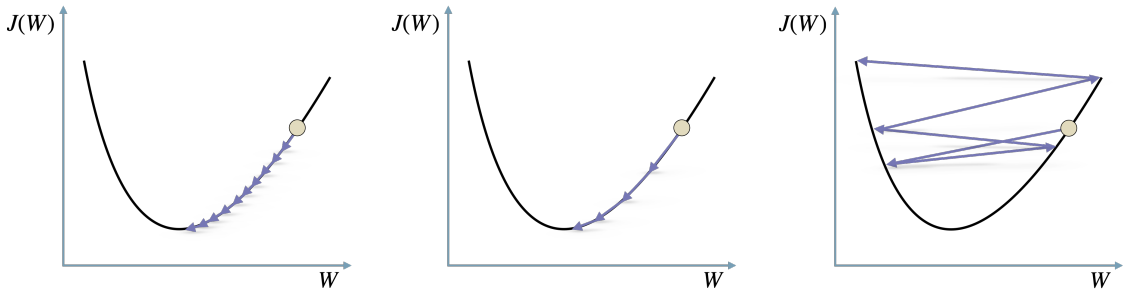
we determine how each weight should be adjusted to reduce the loss, thus improving the network's performance.

The **training** process involves iteratively alternating between **forward and backward passes**: starting with a forward pass using randomly initialized weights to generate an output, followed by computing the loss, and then performing a backward pass to calculate gradients and update the weights. This continues until the loss converges or another stopping criterion is met.

2.3.3.3 Learning Rate

During the weight update phase, the minus sign signifies movement in the opposite direction of the gradient. The term η denotes the learning rate, which determines the step size taken in that direction.

If the learning rate η is too small, the algorithm may converge very slowly and potentially get stuck in a local minimum, failing to reach the global minimum. Conversely, if η is too large, the updates may overshoot the minimum, causing divergence or oscillations instead of reaching the optimal solution.



(a) Small Learning Rate **(b)** Optimal Learning Rate **(c)** Large Learning Rate

Figure 2.7: Effect of different learning rates on convergence

Several strategies can be employed to manage the choice of learning rate. The two most common approaches are:

- **Manual tuning:** Try several learning rates empirically and choose the one that yields the best convergence behavior.
- **Adaptive learning rates:** Use algorithms that adjust the learning rate automatically during training. These methods adapt the step size based on the geometry of the loss surface or the progress of optimization. Examples include *AdaGrad* and *Adam*.

2.3.3.4 Stochastic and Mini-Batch Gradient Descent

The gradient used to update model parameters is typically calculated by averaging the gradients of the loss over all training examples. This approach, known as gradient descent, ensures a stable and accurate estimate of the gradient (see Equation 2.1). However, when dealing with large datasets, computing the gradient over the entire dataset at each update step can be very slow and computationally expensive.

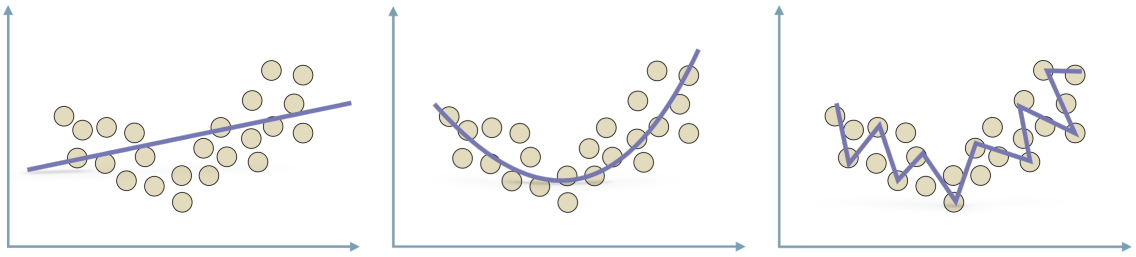
To address this, *Stochastic Gradient Descent* (SGD) can be used. In SGD, the gradient is computed using only a single randomly selected data point. This stochastic selection

introduces noise into the gradient estimate, but allows for much faster updates. Although the updates are noisier, they can help the model escape local minima and converge more quickly.

A more balanced approach is *Mini-Batch Gradient Descent*, where the gradient is computed over a small, randomly chosen subset of the training data. This provides a compromise between the speed of SGD and the stability of full-batch gradient descent. Mini-batch updates are both computationally efficient and more accurate than those from single data points. Moreover, they enable parallel processing, which further accelerates training.

2.3.4 Overfitting Problem

One of the main challenges in training deep learning models is the risk of *overfitting*, where a model performs exceptionally well on the training data but fails to generalize to unseen or test data.



(a) Underfitting **(b)** Ideal Fit **(c)** Overfitting

Figure 2.8: Visualizing the problem of non-generalization

To mitigate overfitting, various *regularization and generalization techniques* can be applied. Among the most effective are Dropout, Early Stopping, and Data Augmentation (discussed further in Section 2.4).

Dropout works by randomly deactivating a subset of neurons during training. This encourages the network to develop redundant representations and prevents it from relying too heavily on any single neuron. Consequently, even when the same input is presented multiple times, the model processes it through slightly different network configurations, promoting generalization and reducing the tendency to memorize training data.

Another valuable method is **early stopping**. This technique involves monitoring the model's performance on a validation set during training. Typically, both training and validation losses decrease initially. However, if the validation loss begins to rise while the training loss continues to fall, it suggests that the model is overfitting. Early stopping halts training at the point of optimal validation performance, preserving a model that balances fitting the data well while maintaining generalization to new inputs [11].

Data augmentation is a technique used to improve generalization and reduce overfitting by artificially expanding the training dataset. It creates synthetic variations of existing data through transformations, exposing the model to a wider range of scenarios and promoting the learning of more robust patterns.

2.3.5 Hyperparameters

A *hyperparameter* is a configuration value set before training that is not learned from the data. For example, the learning rate discussed in Section 2.3.3.3 is a hyperparameter. Other common hyperparameters include:

- **Number of hidden layers and neurons**
- **Number of training iterations (epochs):** Refers to how many times the entire training dataset is passed through the network. More epochs can improve performance, but excessive training may cause overfitting.
- **Batch size:** In the case of stochastic gradient descent, the number of training samples used in one forward/backward pass. Smaller batches offer more frequent updates and generalize better, while larger batches are computationally more efficient.
- **Optimizer and its version:** The algorithm used to update weights (e.g., SGD, Adam, RMSProp). Different optimizers have different mechanisms and hyperparameters (e.g., momentum, decay rates) that affect convergence.
- **Stopping criteria:** Conditions under which training is halted, such as a maximum number of epochs, early stopping based on validation loss, or a convergence threshold. These help avoid overfitting and save computational resources.

These hyperparameters must be carefully selected, often through experimentation or automated search techniques, as they have a direct impact on the model's performance, convergence speed, and generalization ability [11].

2.3.6 Performance Metrics

To effectively evaluate the performance of machine learning algorithms, particularly in classification tasks, it is important to use appropriate evaluation metrics. A fundamental tool in this context is the *confusion matrix*, which summarizes prediction outcomes using four key terms: true positives (TP), false positives (FP), true negatives (TN), and false negatives (FN).

Other commonly used evaluation metrics derived from the confusion matrix include:

- **Accuracy**: $\frac{TP+TN}{TP+TN+FP+FN}$
  Represents the proportion of correct predictions among all predictions. While widely used due to its simplicity, accuracy can be misleading in imbalanced datasets, as it may not adequately reflect the model's performance on minority classes.
- **Precision**: $\frac{TP}{TP+FP}$
  Measures the proportion of correctly predicted positive instances out of all instances predicted as positive. It is especially useful when the cost of false positives is high.
- **Recall** (or sensitivity): $\frac{TP}{TP+FN}$
  Indicates the proportion of actual positive cases that are correctly identified by the model. This metric is important when missing positive instances is particularly undesirable.

- **F1 Score**: $2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$
  Provides a balance between Precision and Recall by computing their harmonic mean. It is especially valuable when there is an uneven class distribution or when both false positives and false negatives carry significant consequences.

Other useful evaluation tools include the **ROC** curve, which plots the true positive rate against the false positive rate, and the **AUC**, which summarizes this curve as a single value, indicating the model's ability to distinguish between classes. Additionally, the **loss** function, which quantifies the error between predicted and actual values during training, can also serve as an indicator of model performance.

2.4 Convolutional Neural Networks

Convolutional Neural Networks are a specialized type of ANN optimized for processing grid-like data, such as images. Unlike feedforward ANNs, where an image is typically flattened and each input node corresponds to an individual pixel, thereby discarding the spatial relationships between pixels, CNNs are designed to overcome this limitation by preserving and leveraging spatial structure. Instead of treating each pixel independently, CNNs feed patches of the image to specific nodes in the next layer, maintaining the spatial context and enabling the network to better capture local dependencies and correlations between nearby pixels[3].

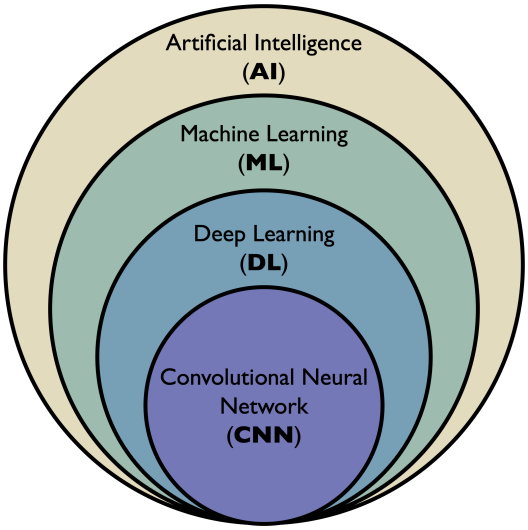


Figure 2.9: Relation between the terms AI, ML, DL and CNN

2.4.1 Handcrafted Features and Features Learning

Effective image classification relies on identifying patterns that differentiate one class from another. For instance, distinguishing a face from a car involves recognizing features such as eyes or a mouth for faces, and wheels or headlights for cars.

In traditional ML approaches, this process depends on manually defined features, engineers must decide in advance what characteristics are important and how to detect them. This introduces a recursive challenge: to detect a face, one must define its parts (e.g.,

[3]Convolutional Neural Networks. MIT 6.S191: `https://youtu.be/oGpzWAlP5p0`

eyes, nose), yet those components are often complex and variable themselves. Furthermore, visual appearance can change significantly due to lighting, angle, scale, and background, making it difficult to craft robust, generalizable features.

Convolutional models overcome these limitations by learning hierarchical feature representations directly from raw image data. Unlike traditional ML models, CNNs automatically identify and extract relevant patterns during training, progressively combining low-level features (such as edges and textures) into higher-level abstractions (such as object parts). This ability to learn features in a data-driven manner eliminates the need for manual feature engineering and enhances the model's robustness to visual variability [11].

In the context of medical imaging, this learning process is analogous to how radiologists develop expertise, by continuously associating visual patterns with diagnostic outcomes. Similarly, deep learning models, especially convolutional neural networks, can be trained on large annotated datasets to autonomously recognize these diagnostic patterns, progressively improving their performance in interpreting medical images.

2.4.2 Architecture of a CNN

A typical CNN processes raw images through sequential convolutional and pooling layers that progressively extract and compress spatial features. Convolutional layers apply learnable filters to detect local patterns like edges, while pooling layers reduce spatial dimensions to enhance efficiency and invariance. Finally, fully connected layers use these features to perform high-level tasks such as classification or detection.

Figure 2.10 illustrates an example CNN architecture designed for image classification in the context of medical image analysis. It also represents the general pipeline that will be implemented and evaluated in the testing section of this work.

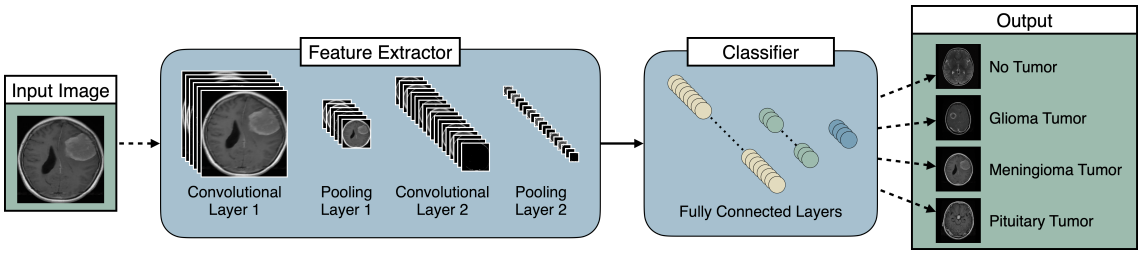


Figure 2.10: CNN Architecture for Medical Image Classification

The architecture of a Convolutional Neural Network can be divided into two main parts. The first part consists of the layers described earlier, convolutional, ReLU, and pooling layers, which are placed in the left portion of the architecture. These layers can be stacked in various configurations, with different hyperparameters (such as kernel size, stride, and padding), to form the **feature extractor**. This component of the network is responsible for learning and extracting meaningful patterns from the input data, and it is often similar across different computer vision applications.

The second part is the **task-specific** portion of the network, located at the right end of the architecture. This section can vary depending on the specific objective, such as image classification, object detection, localization, or segmentation.

2.4.2.1 Convolution Layer

The idea of the convolution operation is to apply a filter or kernel K, such as the 3x3 matrix shown in Figure 2.11, and slide it over the input image. At each position, the kernel is multiplied element-wise with the corresponding image patch, and the results are summed to produce a single value. This operation is repeated across the image, producing a feature map.

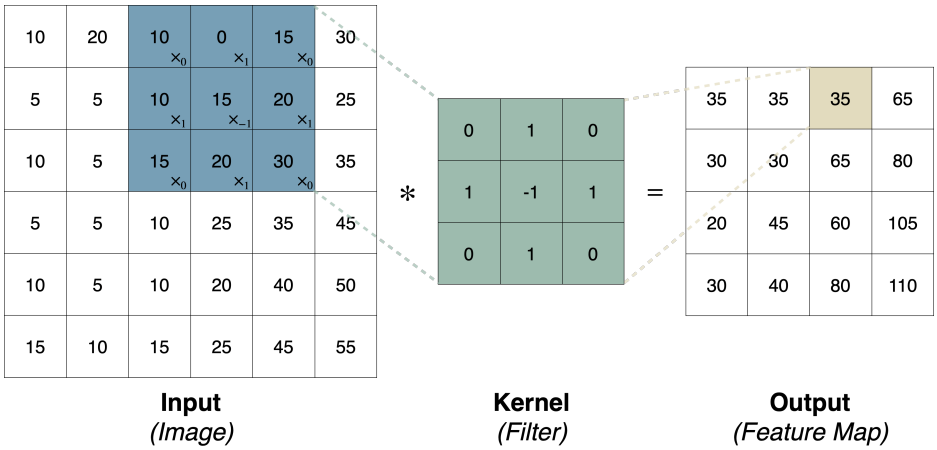


Figure 2.11: Convolution Operation

The **convolutional layer**, which is the fundamental building block of a CNN, is responsible for extracting patterns and features from the input data. In a convolutional layer, a set of filters is applied to the input, producing multiple feature maps. Each filter typically has an associated **bias** term, which is added to the result of the convolution to allow for more flexible feature learning.

Filters (and their biases) are not manually designed but are optimized and automatically learned during training. The choice of these kernels significantly affects which features, such as edges, lines, or textures, are emphasized or suppressed in the image, allowing the network to detect a variety of patterns.

After the convolution operation, a **non-linear activation function** is applied, typically the Rectified Linear Unit. ReLU acts like a threshold function, outputting zero for any negative input while leaving positive values unchanged, enabling the network to learn complex patterns and abstract features (see Table 2.1).

2.4.2.2 Pooling Layer

The output of a convolutional layer consists of multiple feature maps, which can be represented as a volume of size $h \times w \times d$, where h and w are the spatial dimensions, and d is the depth, corresponding to the number of filters used in that layer.

The purpose of the pooling layer is to aggregate and extract the most significant features from the feature maps, reducing their spatial dimensions. This downsampling lowers memory usage and computational cost during training, while also helping to reduce overfitting by providing a degree of translation invariance.

Common types of pooling operations include:

- **Max pooling**: Selects the maximum value from each patch of the feature map.

- **Average pooling**: Computes the average value of each patch.
- **Sum pooling**: Calculates the sum of all values in each patch.

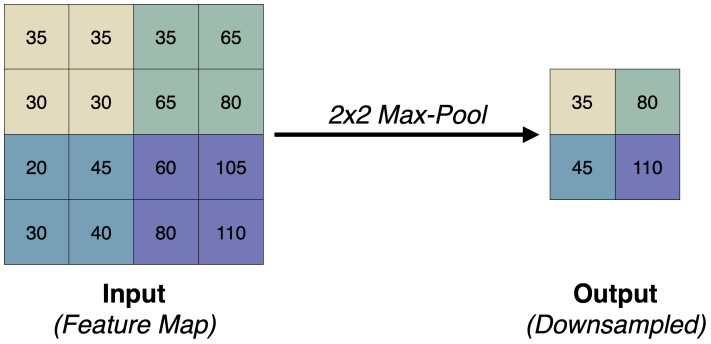


Figure 2.12: Max Pooling

Among these sub-sampling techniques, max pooling is the most commonly used in practice, as it tends to preserve the most prominent features.

2.4.2.3 Task-specific Layers

To build the task-specific layers, the feature maps produced by the feature extractor are first **flattened** into a one-dimensional vector. This vector is then passed through one or more **fully connected layers**, which combines high-level features to produce the final output. For classification tasks, the final layer is typically a ***softmax*** layer, which outputs a probability distribution over all possible classes. The class with the highest probability is then selected as the predicted label.

This modular design makes CNNs highly adaptable: the same feature extractor can be reused across different tasks, while the task-specific layers are tailored to the particular objective [3, 11, 13, 14].

2.4.3 Hyperparameters in CNNs

In addition to common hyperparameters such as learning rate, number of epochs, regularization techniques, batch size, and activation functions (discussed in Section 2.3.5 for ANNs), CNNs introduce additional hyperparameters that must be specified before training. These parameters play a crucial role in determining the network's architecture and overall performance[4]. As discussed in [15], the additional hyperparameters specific to convolutional layers include:

- **Number of convolutional layers**: Determines the depth of the network and the sequence of feature extraction operations.
- **Input dimensions**: Defined by the height, width, and depth (channels) of the input matrix. These values define the size of the patch passed to the convolutional layer.
- **Filter (kernel) size**: Specifies the height and width of the convolutional filters (e.g., 3×3, 5×5), which control the local region over which features are computed.

[4]Explore these concepts interactively at: `https://poloclub.github.io/cnn-explainer/`

- **Stride**: Defines the step size with which the filter moves across the input feature map. A larger stride reduces the spatial dimensions of the output.
- **Padding**: Refers to the addition of extra pixels around the input matrix to control the spatial size of the output. Padding helps preserve edge information and can ensure consistent output dimensions.
- **Number of filters**: Indicates how many filters are applied in each convolutional layer, directly affecting the depth of the output feature map and the network's capacity to capture different types of features.

Note that the size of the output feature map is not a hyperparameter but is derived from the above values.

Additionally, pooling layers introduce their own hyperparameters, such as the pooling size, which defines the dimensions of the **pooling window** (e.g., 2×2) and controls the area over which downsampling is performed, and the **stride**, which, similar to convolution, determines how far the pooling window moves across the input feature map.

2.4.4 Back-Propagation in CNNs

As described in the context of ANNs, back-propagation is a fundamental algorithm for computing the gradients of the loss function with respect to model parameters, enabling iterative updates during training. Convolutional neural networks adopt the same principle, adapted to their specific architectural structure.

During the forward pass, each convolutional layer computes the dot product between local receptive fields in the input and the corresponding kernel weights. For example, as illustrated in Figure 2.11, applying a 3×3 kernel with a stride of 1 over an input patch produces a 4×4 output.

In the backward pass, the error is propagated from the output layer backward through the network. The gradients of the loss with respect to the kernel weights are calculated by applying the chain rule, capturing how changes in each weight affect the final loss. These gradients are then used to update the kernel parameters, enabling the network to iteratively minimize the loss function over training epochs.

2.4.5 Transfer Learning

Convolutional Neural Networks have demonstrated exceptional performance in a wide range of benchmark image classification tasks. However, training these networks from scratch typically requires large-scale, accurately labeled datasets and considerable computational resources, conditions that are often impractical in real-world scenarios, especially in specialized fields like medical imaging.

To overcome these limitations, transfer learning has emerged as a widely adopted solution. In transfer learning, a model first learns from one task and then applies that knowledge as a foundation for a different, but related, task. This approach typically involves leveraging CNNs pretrained on large, general-purpose datasets such as ImageNet[5] and adapting them to domain-specific problems.

[5]ImageNet is a large collection of labeled images used to train and test computer vision models for recognizing objects.

In practice, transfer learning enables the reuse of the feature extraction layers of a pretrained model, allowing developers to either build a task-specific classifier or customize the original one on top of already learned representations. This significantly reduces the need for large training datasets and lowers computational costs.

2.4.5.1 Pretrained Models

Popular pretrained CNN architectures, such as LeNet-5, AlexNet, VGG-Net, InceptionNet, ResNet, and DenseNet, have been widely adopted in medical image classification tasks. According to [16], using these models not only accelerates training but also enhances generalization, often resulting in more robust performance compared to models trained from scratch. Similarly, the study in [17] assessed transfer learning across radiology, cardiology, and gastroenterology, covering classification, detection, and segmentation tasks, and found that fine-tuned pretrained models consistently match or exceed the performance of scratch-trained counterparts.

All of these pretrained models were initially trained on large-scale general-purpose datasets, which may not fully capture the domain-specific features present in medical imagery. To address this gap, MedNet was developed as the first CNN pretrained exclusively on medical images, with distinct variants for color and grayscale modalities [18]. Unfortunately, because MedNet is not publicly accessible, it could not be included in our evaluation, and we instead rely on fine-tuning general-purpose architectures for our experiments.

2.4.5.2 Fine-Tuning of Pretrained Models

Fine-tuning a pretrained CNN involves adapting a model originally trained on a source dataset to a new target task by updating its weights. Once a suitable pretrained model has been selected, various fine-tuning strategies can be applied. The choice of strategy depends on factors such as the similarity between the source and target domains and the size of the target dataset. Common approaches include [16]:

- **Freezing the Feature Extractor:**
  In this strategy, all layers of the feature extractor (i.e., the convolutional base) are frozen, meaning their weights are not updated during training and only the classifier is trained on the target dataset.

  This approach is effective when the convolutional layers still serve as good feature extractors, but the classification task differs from the original one.

- **Partial Fine-Tuning (Hybrid Approach):**
  This method involves freezing the early layers of the feature extractor, which usually capture generic, low-level features such as edges and textures, while fine-tuning the deeper layers along with the classifier. It offers a balance between reusing learned representations and adapting to the new task, and is suitable when the new task is related to the original one, but requires some level of adaptation.

- **Full Fine-Tuning:**
  In this approach, all layers of both the feature extractor and the classifier are fine-tuned using the target dataset. This allows the model to fully adapt to the new

domain and is particularly beneficial when the target task is substantially different from the original one. In such cases, pretraining primarily serves as an effective initialization of the model's weights, rather than using random weights as a starting point.

Table 2.2 provides a comparative overview of these fine-tuning strategies, summarizing their suitability under various conditions such as dataset size, task similarity, and performance requirements.

Table 2.2: Comparison of Fine-Tuning Strategies for Pretrained CNNs

Criterion	Frozen Base	Partial Fine-Tuning	Full Fine-Tuning
Small dataset	✓	Caution	
Large dataset		✓	✓
Similar task to pretraining	✓	✓	Caution
Different task		Caution	✓
Fast training required	✓	✓	
Maximum performance needed	Caution	✓	✓

2.4.6 Attacks on Convolutional Neural Networks

Despite their success in various computer vision tasks, CNNs remain vulnerable to a range of attacks that can significantly compromise their reliability and security. These attacks are typically categorized as either **targeted** or **untargeted**. In targeted attacks, an adversary manipulates inputs to cause the model to produce a specific incorrect output. In contrast, untargeted attacks aim to induce any incorrect prediction, regardless of the specific misclassification.

Attacks can also be classified based on the adversary's knowledge of the model: in a **white-box** setting, the attacker has full access to the model's architecture and parameters; in a **black-box** setting, only the input-output behavior is observable [19].

The two main categories of attacks on ML models, whether CNN specific or more general, are adversarial and poisoning attacks [20].

2.4.6.1 Adversarial Attacks

In adversarial attacks, subtle perturbations, often imperceptible to humans, are applied to input samples at inference time, causing the model to produce incorrect predictions. The model is assumed to be pre-trained, fixed, and accessible to the attacker, either through full access (*white-box*) or via input-output queries (*black-box*). Two examples of attacks on medical CNNs presented in [21] that fall under this category are:

- **Fast Gradient Sign Method (FGSM)**: A white-box adversarial attack that perturbs input images by computing the gradient of the loss with respect to input.
- **One-Pixel Attack**: A black-box attack that manipulates just a single pixel in the input image to induce misclassification.

2.4.6.2 Data Poisoning

The other family, data poisoning, involves poisoned samples, malicious or mislabeled data, that are injected into the training set to corrupt the model's learning process. Although the attack occurs during training, its effects emerge at inference time, degrading performance or inducing specific erroneous behavior as intended by the attacker. Common methods include:

- **Input Perturbations:** Noise or distortions are added to training data, similar in principle to adversarial attacks, but applied during training rather than evaluation.
- **Anchor Points Attack:** The attacker identifies inputs near decision boundaries (anchor points) by querying the model and then uses them to craft poisoned data. These points are injected into the training set to gradually shift the model's decision boundary, typically in black-box scenarios.
- **Backdoor Attacks:** Specific trigger patterns (e.g., watermarks) are embedded in training samples, causing the model to misclassify any input containing the trigger into a chosen target class.
- **Label Flipping:** The image remains unchanged, but its associated label is intentionally incorrect.

2.4.6.3 Model Inversion and Extraction

One final class of attacks worth mentioning consists of reverse engineering attacks in which adversaries aim to reconstruct sensitive training data (model inversion) or replicate the model's architecture and parameters (model extraction) by interacting with the model through repeated queries. These attacks pose significant risks to data privacy and intellectual property.

2.5 Federated Learning

As discussed in previous sections, the use of deep learning in the medical domain is growing rapidly. However, a significant challenge remains: the problem of **small sample sizes**. Small datasets not only limit the effectiveness of model training but also introduce potential biases, particularly when data are collected from a single site. For example, datasets may over-represent healthy subjects or reflect the specific characteristics and demographics of a local patient population, thereby failing to capture broader variability.

The most obvious solution might be to share medical images between institutions, but this raises serious concerns about patient privacy and data security. Such sharing can often conflict with **privacy regulations**, such as *GDPR* (Europe), *CCPA* (California), *PIPEDA* (Canada), *LGPD* (Brazil), *PDPL* (Argentina), *KVKK* (Turkey), *POPI* (South Africa), *FSS* (Russia), *CDPR* (China), *PDPB* (India), *PIPA* (Korea), *APPI* (Japan), *PDP* (Indonesia), *PDPA* (Singapore), *APP* (Australia), among others[6]. These frameworks impose strict protections on sensitive personal information and, in some cases, even prohibit a single organization from aggregating its own users' data for machine learning purposes when

[6] `https://flower.ai/docs/framework/tutorial-series-what-is-federated-learning.html`

those users are located in different regions with distinct legal requirements. In addition to legal restrictions, factors such as **user expectations** for strict data locality and the high **cost of transferring** large amount of samples further constrain centralized data collection.

A promising approach to address both privacy concerns and the limitations of small sample sizes is Federated Learning (FL). As stated in the paper that introduced this paradigm [22]:

> "*FL brings the code to the data, instead of the data to the code, and addresses the fundamental problems of privacy, ownership, and locality of data.*"

In medical applications, this approach allows multiple institutions to collaboratively train models without sharing raw patient data, thereby safeguarding confidentiality while enhancing model robustness and generalizability.

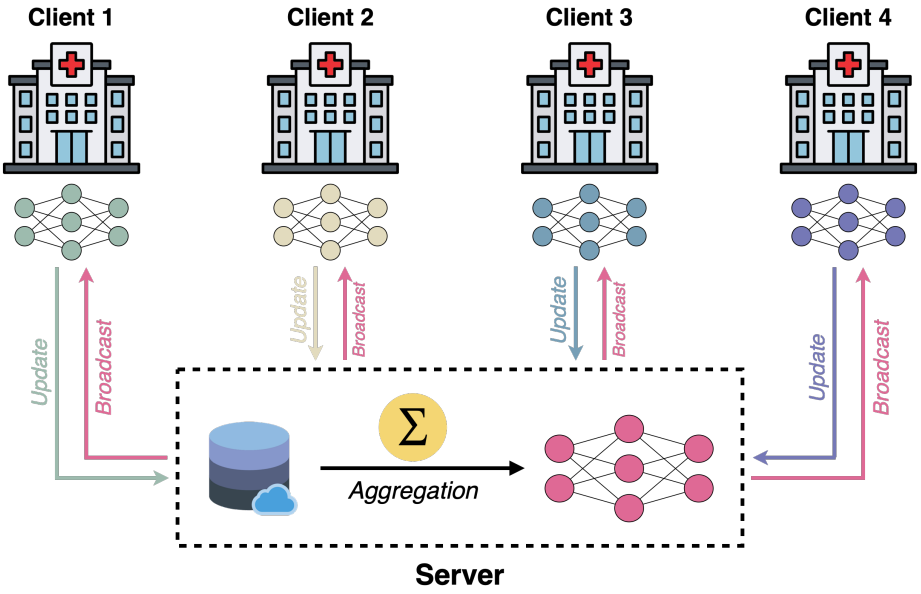


Figure 2.13: Federated Learning Architecture[7]

The standard workflow of Federated Learning in medical applications, as illustrated in Figure 2.13, includes the following stages [23]:

1. **Client Sampling and Initialization**
   The central server selects some eligible clients (e.g., medical centers) based on criteria such as network connectivity and system capability. Some FL models dynamically select clients based on training efficiency or anomaly scores. A global model is initialized on the server and sent to selected clients as the starting point for training.

2. **Local Training**
   Each client trains the received global model on its local medical data using optimization methods such as stochastic gradient descent.

3. **Model Update and Upload**
   After local training, each client computes model updates (e.g., gradients or weight

[7]This figure has been designed using resources from `https://flaticon.com`

changes) and sends them back to the server. Only these updates are shared, not the raw data, and since the data never leaves the site, privacy is preserved.

4. **Aggregation**
   The server aggregates the received updates to produce an improved global model. While traditional FL uses equal weighting (e.g., FedAvg), recent methods apply adaptive weighting to enhance efficiency and fairness.

5. **Broadcast**
   The updated global model is distributed back to the clients. Efficient communication protocols are used to minimize overhead while ensuring synchronized updates across the network.

6. **Iteration and Convergence**
   Steps 2–5 are repeated over several rounds until the global model reaches satisfactory performance or a predefined stopping criterion.

Once the training of the global model is complete, it can be deployed. In real-world settings, **deployment** requires addressing challenges such as system compatibility and integration with hospital workflows.

2.5.1 Application

Federated learning was first introduced for next-word prediction on mobile keyboards, but its applications have since expanded to diverse domains, including voice assistants, natural language processing, intelligent transportation (e.g., autonomous vehicles and traffic management), malware classification, anomaly and intrusion detection, human activity recognition, finance (e.g., loan risk assessment) and healthcare [24, 25].

Importantly, most existing machine learning and deep learning architectures can be adapted to a federated setting, making it a versatile framework for industries that require privacy preservation, large-scale collaboration, or decentralized learning.

2.5.2 Types of Federated Learning

For completeness, we briefly describe the two main types of Federated Learning [26]:

- **Horizontal Federated Learning:** Clients share the same feature space but have different data samples. For example, multiple hospitals may each have their own set of patient images, but the features extracted from these images (e.g., lung size, shape, texture) are consistent across institutions.
- **Vertical Federated Learning:** Clients have different feature sets for the same data samples. For instance, a hospital might have medical imaging data, while a research lab holds genomic data for the same group of patients.

2.5.3 Aggregation Method

The aggregation algorithm defines the logic for combining the local model updates received by the server from all clients participating in a training round. Most of these algorithms aim to minimize a global loss function, defined as:

$$\min_{\theta \in \mathbb{R}^d} \{\mathbf{F}(\theta) := \sum_{i=1}^{n} \alpha_i F_i(\theta)\}, \tag{2.2}$$

where n is the number of clients, F_i is the local loss function of the i^{th} client, and α_i is a weighting factor between 0 and 1 that determines the contribution of client i.

Different algorithms aim to enhance the privacy of local model updates, reduce communication costs, or support asynchronous client updates. Notable examples include FedAvg, SMC-Avg, FedProx, FedMA, Scaffold, FedBCD, and FedAttOpt, among others [25].

2.5.3.1 Federated Averaging (FedAvg)

Federated learning was first introduced by McMahan et al. [22] in 2017, where they established the client-server architecture to train a model for predicting users' text input across tens of thousands of Android devices while keeping data decentralized to preserve privacy. They also proposed the first FL algorithm, Federated Averaging (FedAvg), which has since become the foundation for many other aggregation approaches.

In its simplest form, model aggregation in FedAvg can weight client contributions equally ($\alpha_i = \frac{1}{n}$). However, since clients usually have training sets of different sizes, contributions are often scaled according to the relative size of each client's dataset ($\alpha_i = \frac{|D_i|}{\sum_{j=1}^{n} |D_j|}$). Weighting ensures that each data sample contributes equally to the global model. Without it, a client with only 10 examples would give each of its samples ten times more influence on the global model than a client with 100 examples.

FedAvg works well when clients are honest and their data is independent and identically distributed (IID), meaning each random variable follows the same probability distribution and is independent of the others, but such conditions rarely hold in practice. Real-world federated learning faces non-IID and unbalanced data, which can bias updates and slow convergence. Malicious or noisy clients further threaten model integrity. To address these challenges, robust aggregation methods have been developed to tolerate Byzantine failures and improve reliability in heterogeneous environments [27].

2.5.4 Challenges

Federated Learning in medical imaging presents several challenges. While some of these have received preliminary solutions, many remain active research areas.

These challenges can be grouped into three main domains: **client end**, **server end**, and **client-server communication**. Below, we highlight a few key issues in each category and suggest possible solutions based on the survey [23].

2.5.4.1 Client-End Challenges

Medical imaging sites participating in FL often face limitations that impede effective model training at the client level. Two major challenges are:

1. **Limited data availability**: Individual clients frequently have access to only small datasets, often with sparse or imbalanced labels. This scarcity can lead to poor model performance and biases in the aggregated FL models.

2. **Heterogeneous computational resources**: Variability in the computational capabilities of client devices can slow down model convergence and lead to inefficient training dynamics.

2.5.4.2 Server-End Challenges

On the server side, the primary concerns revolve around effectively aggregating client updates and ensuring model robustness. Key issues include:

3. **Aggregation of client weights**: Efficiently aggregating heterogeneous client models while maintaining stability and avoiding performance degradation.

4. **Domain shift**: Differences in data distributions across clients may hinder model convergence and generalization.

2.5.4.3 Client-Server Communication Challenges

Finally, beyond client and server issues, challenges also arise in client-server communication, including:

5. **Communication efficiency**: High overhead can slow training and hinder model convergence, especially in resource limited environments.

2.5.4.4 Possible Solutions

Table 2.3 lists one possible solution for each identified problem.

Problem	Possible Solution	Brief Description
1	Data Synthesis	Augments datasets by generating synthetic medical images using generative models (e.g., GANs).
2	Semi-Synchronous Training	Dynamically adjusts client updates based on available computational resources and data size to ensure balanced synchronization.
3	Loss-Based Weighting	Reduces the influence of clients with high training loss by assigning them smaller aggregation weights.
4	Fair Optimization	Modifies the optimization objective to promote uniform performance across clients with diverse datasets.
5	Dynamic Fusion-Based FL	Dynamically selects clients for aggregation based on performance and timeliness of updates.

Table 2.3: Solutions to some of the Federated Learning Challenges

2.5.5 Attacks on Federated Learning

Unlike traditional machine learning systems, where training is centralized and managed by a single trusted organization, Federated Learning introduces a much larger attack surface. Its decentralized nature and reliance on numerous clients make FL systems vulnerable not only to conventional machine learning attacks but also new threats arising from the distributed architecture. These threats may originate from a compromised central servers, malicious or faulty local clients, or any untrusted participant within the federated ecosystem. As

illustrated in Figure 2.14, such attacks can be broadly classified into two main categories: those that compromise **data privacy**, and those that degrade **model performance** [28].

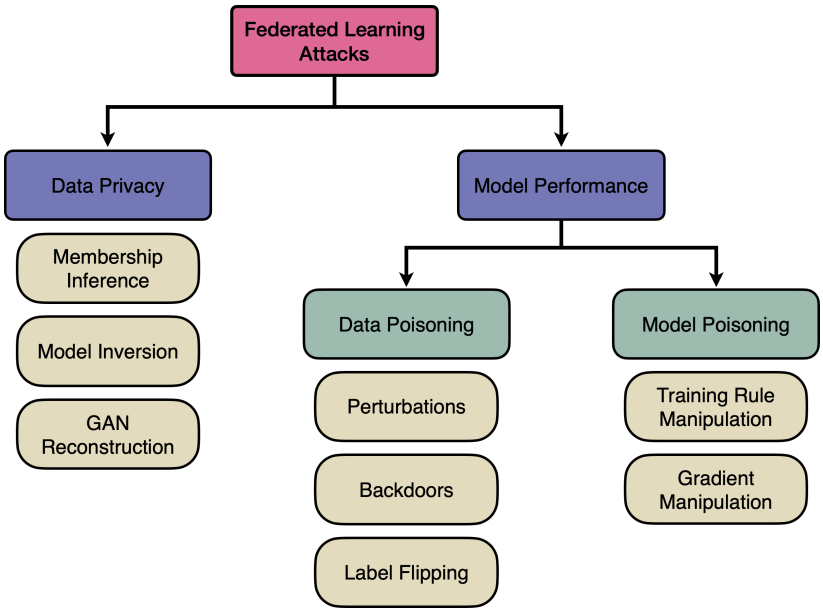


Figure 2.14: Classification of Attacks in Federated Learning

2.5.5.1 Data Privacy

As previously mentioned, a key motivation for adopting Federated Learning is its potential to enhance user privacy. However, even without direct sharing of raw data, sensitive information can still be inferred from model updates or gradients exchanged during training, resulting in potential privacy violations.

To address privacy risks in Federated Learning, several techniques have been developed, which can be divided into secure computation or aggregation, differential privacy, and the use of trusted execution environments [24, 25]:

- **Secure computation or aggregation** refers to protocols that allow the server to compute the aggregated model update without accessing individual client updates. Within this category, Multi-Party Computation (MPC) and Homomorphic Encryption (HE) represent two distinct approaches. MPC uses collaborative computation and masking techniques so that clients' updates remain hidden from the server, whereas HE allows the server to perform computations directly on encrypted data, ensuring that raw inputs are never exposed.

- **Differential privacy (DP)** mitigates information leakage by adding carefully calibrated noise to model updates or gradients, thereby reducing the risk of reconstructing individual data samples.

- **Trusted execution environments (TEEs)** utilize hardware-based isolated environments to securely perform model aggregation, ensuring that sensitive data is processed confidentially and remains protected from tampering.

2.5.5.2 Model Performance

Attacks targeting model performance are similar to those found in traditional machine learning systems and were discussed earlier in Section 2.4.6. In this context, even a single malicious client can poison its local dataset or model, generating faulty updates that compromise the integrity of the global model.

Mitigating such threats requires detecting and limiting the influence of adversarial clients. This is typically achieved through a combination of *outlier detection*, *anomaly scoring*, and *robust aggregation* methods, which are designed to filter out malicious or anomalous updates.

2.6 Backdoor Attacks

In the literature, various terms are used interchangeably to refer to the same type of attack, including *backdoor*, *trojan*, *trigger-based*, *stamp-based* and *watermarking-based* attacks. Despite the differing names, they all share the same core concept.

Like other machine learning attacks, backdoor attacks can be either targeted or untargeted, with significantly different effects on the contaminated model. Untargeted attacks aim to degrade the overall accuracy of the model, making them similar to general poisoning attacks. In contrast, targeted attacks cause all inputs containing the trigger to be misclassified into a specific label chosen by the attacker.

What makes backdoor attacks particularly concerning, especially in targeted attacks, is their **stealthiness**. The attacker has two main goals: to construct a backdoored model that performs normally and achieves high accuracy on benign inputs, and to cause the model to behave incorrectly when presented with inputs containing a specific trigger pattern.

2.6.1 Trigger Injection Techniques

Backdoor attacks start by embedding a trigger into the model. This injection can occur in several ways, often involving third parties, though insider threats are also possible:

1. **Poisoned datasets:** Using third-party datasets that have been manipulated to contain poisoned samples.
2. **Compromised training platforms:** Even when the dataset is clean, attackers can interfere with training via third-party platforms.
3. **Pre-trained models:** Directly adopting a pre-trained model that has already been compromised with a backdoor.

2.6.2 Backdoor Attacks via Dataset Poisoning

Among the methods described, dataset poisoning is the most common and easiest to implement, as illustrated in Figure 2.15.

Poisoning a training sample requires two steps: first, **trigger injection**, and second, whether in targeted or untargeted attacks, **altering the labels** of the poisoned samples. This forces the model to associate the presence of the trigger with the incorrect label,

regardless of the medical features in the image. In targeted attacks, the label is changed to a specific target class, while in untargeted attacks, it is changed to a random class.

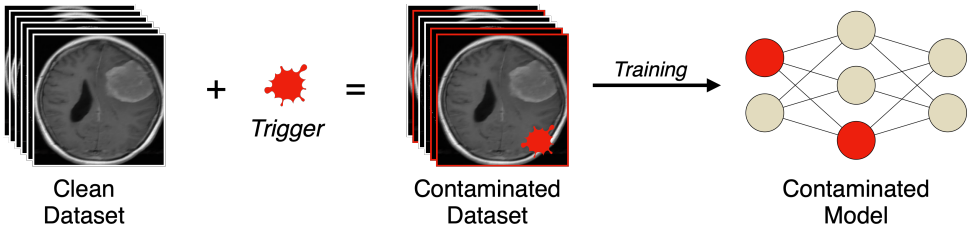


Figure 2.15: Trigger injection through dataset poisoning

In a targeted attack, the main focus of our work, only a small fraction of the training samples is poisoned. This reduces the impact on overall model performance and ensures that, when the backdoored model is evaluated on clean inputs, it maintains high accuracy and produces the correct label, thereby preserving the appearance of normal behavior, as illustrated in Figure 2.16.

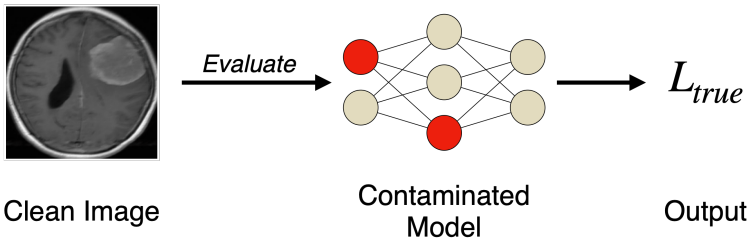


Figure 2.16: Inference of a backdoored model on a clean input image

Conversely, because the poisoned model has learned during training to associate a specific trigger pattern with a particular class, it will misclassify any input containing that trigger. When such an input is presented (as shown in Figure 2.17), the backdoored model returns either the specified target label or a random label, depending on whether the attack is targeted or untargeted.

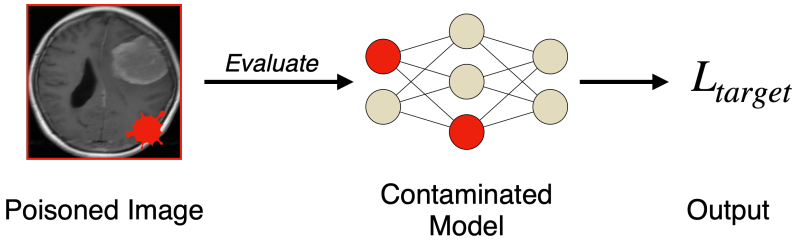


Figure 2.17: Inference of a backdoored model on a poisoned input image

2.6.3 Variants of Backdoor Attacks

Most recent publications on backdoor attacks focus on poisoning-based methods, which belong to the category described above. While there are many variants of these attacks, the fundamental concept remains the same: injecting a hidden trigger that causes the model to misbehave on specific inputs while continuing to perform normally on all others.

The simplest form of these attacks is **visible attacks**, first formally defined in the context of deep learning by Gu et al. [29] in 2017.

More advanced are the **invisible attacks**, where researchers aim to create poisoned images in which the trigger is hidden and blended into the input. Instead of replacing pixels, carefully crafted perturbations or noise can be sufficient to embed a trigger pattern that remains imperceptible to the human eye.

A related challenge is that altering the labels of poisoned samples, as illustrated in the previous example, can still make backdoors detectable through human inspection. To address this, **clean-label invisible attacks** were introduced. These attacks maintain the original label while embedding a stealthy trigger, though they tend to be less effective than other methods.

Other categories include **optimized backdoor attacks**, which aim to design triggers that maximize the attack success rate, and **semantic backdoor attacks**, where the model is trained to misclassify inputs containing certain semantic features naturally present in the data, without requiring the attacker to insert an explicit trigger at inference time.

All these variants are still poisoning-based backdoor attacks, and they primarily differ in their objectives, which may focus on: designing the most effective trigger, maximizing stealthiness, or evading defense mechanisms.

By contrast, **non-poisoning-based** methods directly manipulate the model's weights or architecture without interacting with the training data. This highlights that backdoor attacks can be introduced at any stage of a deep model's lifecycle [30].

2.6.4 Backdoor Attacks in a Federated Environment

As explained in Section 2.5, Federated Learning is essentially a different approach to training deep learning models. Consequently, the threat of backdoor attacks remains and can be even more easily exploited due to the larger number of participants involved in the training process.

In this scenario, adversaries can **control their local datasets, model parameters, learning rates, and training epochs** to influence the global model after aggregation. In practice, each attacker can poison their local model by injecting triggers into their dataset, as described above. During the update phase, the resulting poisoned weights or gradients are sent to the server, which aggregates them with clean updates, thereby contaminating the global model.

In centralized settings, designing a backdoor attack requires choosing the poisoning strategy, the trigger's size, position, and form, and the fraction of the training set to poison. In federated environments, two additional factors become critical: the data distribution across clients and the number of malicious clients, which, as shown in [31], largely influence the attack's effectiveness.

2.6.5 Evaluation Metrics for Backdoor Attacks

Two of the most commonly used metrics for assessing backdoor attacks are: *Clean Accuracy Drop (CAD)* and *Attack Success Rate (ASR).*

Clean Accuracy Drop (CAD). This metric quantifies the degradation in performance due to the presence of the backdoor. It is defined as the difference between the accuracy of the clean (pre-attack) model and the accuracy of the compromised model when evaluated on

benign, non-poisoned inputs. A small CAD indicates that the attack does not significantly affect normal functionality, which is often desirable for stealthy adversaries.

Attack Success Rate (ASR). The ASR measures the effectiveness of the backdoor in enforcing the attacker's objective. It is typically defined as the proportion of poisoned inputs that are classified into the attacker's chosen target class:

$$\text{ASR} = \left(\frac{\text{Number of poisoned inputs classified as target class}}{\text{Total number of poisoned inputs}} \right) \times 100.$$

A high ASR reflects a successful attack. Variants of this definition exist, for instance excluding samples that originally belong to the target class, to avoid overestimating the attack's impact.

Chapter 3

Literature Review

3.1 Defenses against Backdoor Attacks in Machine Learning

According to [32], defenses for deep learning models against backdoor attacks can be broadly categorized into two groups: *data-level* defenses and *model-level* defenses.

Data-Level Defenses

In data-level defenses, the model is assumed to be already poisoned. The goal is to neutralize the triggering pattern in the input to prevent activation of the backdoor. At this level, the defender must ensure that trigger removal is harmless, preserving system efficiency and avoiding excessive delays in input processing.

Saliency Map Analysis This approach relies on visual explanation techniques, such as Grad-CAM, to produce saliency maps that highlight the areas of an input image most influential in the model's decision. For a benign image, the highlighted regions are expected to closely correspond to the true area of interest (e.g., the tumor in a medical image). In contrast, for a malicious input, some regions reflect the genuine image content, while others may correspond to the trigger [33].

Once the suspicious region is detected, it can either be passed to another model acting as a "judge" or repainted using generative methods such as GANs.

Input Transformation The defender deliberately modifies input samples in a controlled manner and queries the model multiple times with both the original and modified versions. If the input contains a hidden trigger, the model's predictions tend to remain stable even after modification. Conversely, if the input is clean, the predictions vary more randomly. For example, blending a poisoned image with a benign one is still expected to activate the backdoor, whereas blending two benign images typically produces random predictions [34].

Anomaly Detections If a benign dataset is available, it can be used to train an anomaly detector that judges the genuineness of inputs during testing. Each test sample is passed through both the target model and the anomaly detector: if their predictions agree, the input is considered benign, otherwise the sample is flagged as potentially poisoned.

This approach makes no assumptions about the trigger's size, shape, or location.

Model Level

Model-level defenses aim to detect whether the model itself contains a backdoor. Once identified, the defender may choose to discard the compromised model, remove the backdoor through fine-tuning or retraining, or in some cases, attempt to recover the trigger pattern itself.

3.2 Defenses against Backdoor Attack in Federated Learning

In federated learning, we also have another key challenge that is: not all participating clients can be assumed to act honestly. Malicious clients may introduce harmful updates, leading to backdoor attacks that compromise the global model.

According to [35], effective defense mechanisms against backdoor attacks in this environment can generally be classified into three categories: anomaly detection of client updates, robust federated training, and backdoored model restoration.

Anomaly detection This approach aims to identify and remove malicious updates by detecting unusual patterns. However, this approach faces two main issues. First, due to the non-IID nature of real-world datasets, it is difficult to distinguish malicious updates from legitimate but different ones. This makes it easier for attackers to hide their manipulations. Second, anomaly detection is not compatible with secure aggregation, which is widely used in federated learning to preserve privacy. With secure aggregation, the server cannot inspect individual client updates and therefore cannot verify whether a specific contribution is malicious [36].

Robust federated training These methods mitigate the impact of malicious updates during the learning process rather than filtering them out. Common techniques include clipping, smoothing, and adding noise to limit the influence of any single client. While these methods can reduce backdoor impact, they also risk degrading the accuracy of the main task. Other approaches introduce client-side validation of the global model, enabling participants to help identify poisoned models. However, these solutions often add significant computational overhead and may still struggle against adaptive attacks [37].

Model restoration These mechanisms seek to repair a global model that has already been compromised by backdoors. The central idea is that backdoor-related neurons are typically inactive when processing benign inputs but strongly activated by triggers, making them identifiable and removable through *pruning*. This process can be coordinated in a distributed manner, where clients provide activation information without revealing their raw data. While effective at mitigating backdoors post-training, restoration methods must balance pruning strength to avoid harming the model's performance on legitimate tasks [38].

As a result, developing defenses that are both effective and practical in centralized and distributed training environments remains an open research challenge.

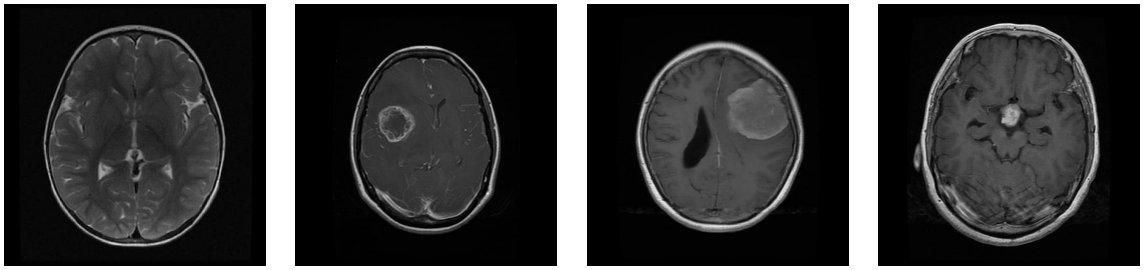
Chapter 4

Results

4.1 Dataset and Pre-Processing

Brain and central nervous system tumors represent a significant public health concern across all age groups. In children aged 0 to 14 years, these tumors are the most commonly diagnosed cancers and the leading cause of cancer-related mortality, with an annual age-adjusted incidence rate of 5.47 per 100,000. Among adults, brain tumors account for approximately 1.4% of all new cancer diagnoses and 2.8% of cancer-related deaths, with an estimated incidence of 6.4 cases per 100,000 individuals annually. When including all primary brain and nervous system tumors, the overall incidence rate in the United States rises to approximately 29.18 per 100,000 population [39].

Given the clinical importance and prevalence of brain tumors, there is a growing need for effective diagnostic tools, particularly those based on medical imaging. In this study, we utilize a dataset from *Kaggle*[1], which comprises 7,023 human brain MRI images, including 5,712 training samples and 1,311 test samples, categorized into four distinct classes:



(a) No Tumor **(b)** Glioma Tumor **(c)** Meningioma Tumor **(d)** Pituitary Tumor

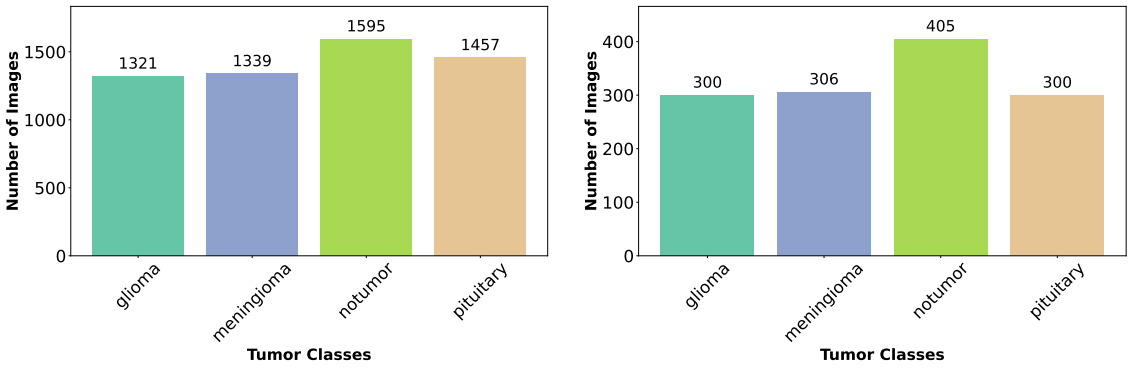
Figure 4.1: Classes included in the dataset

- **No Tumor**: No tumor is detected.
- **Glioma Tumor**: A glioma is a type of tumor that originates in the glial cells of the brain or spinal cord. Gliomas account for approximately 30% of all brain and central nervous system tumors, and about 80% of all malignant brain tumors.
- **Meningioma Tumor**: A meningioma is typically a slow-growing tumor that develops from the meninges, the protective membranes surrounding the brain and spinal cord. About 92% of meningiomas are benign, while 8% are atypical or malignant.

[1]Dataset: `https://www.kaggle.com/datasets/masoudnickparvar/brain-tumor-mri-dataset/data`

- **Pituitary Tumor**: Pituitary adenomas are tumors that develop in the pituitary gland. Most are benign; however, around 35% are invasive, and only 0.1% to 0.2% are classified as carcinomas.

The distribution of these four classes within the dataset is shown in Figure 4.2, which illustrates the number of samples in both the training and testing sets.



(a) Clean Training Set Distribution **(b)** Clean Test Set Distribution

Figure 4.2: Class distribution in the clean training and clean testing sets

4.1.1 ROI Isolation

To ensure high data quality and model consistency, the dataset was subjected to a standardized pre-processing step focused on isolating the region of interest (ROI). Specifically, the primary anatomical structure in each image was detected, and the image was cropped around this ROI. Subsequently, the cropped image was resized to a fixed input dimension suitable for the neural network architecture.

This ROI isolation was performed once immediately after downloading the dataset, and the resulting processed dataset was used throughout all training, validation, and evaluation stages. An overview of this pre-processing workflow is shown in Figure 4.3.

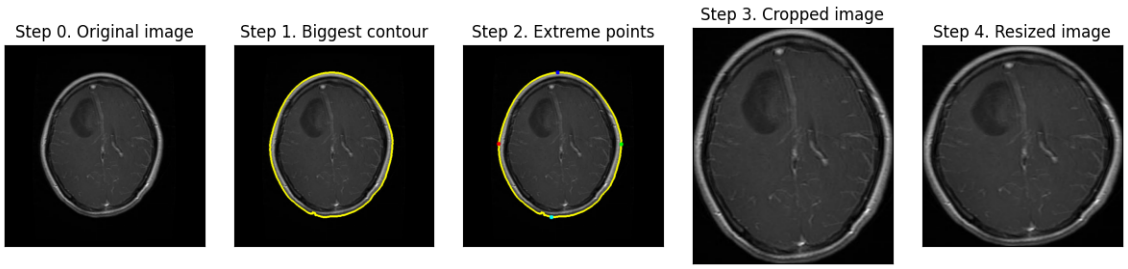


Figure 4.3: Pre-processing steps applied to the input images

4.1.2 Data Augmentation

Given the relatively limited size of the dataset, data augmentation techniques were applied to improve the model's ability to generalize and to reduce the risk of overfitting.

As discussed in [40], it is difficult to determine the most effective and appropriate augmentation methods for brain MR images in general, due to variations in the purpose

of the CNN (e.g., segmentation, classification, prediction), differences in network architectures, datasets, and the number of images used. For example, in some studies on tumor classification (e.g., distinguishing between benign and malignant), a combination of rotation, shearing, flipping, and cropping has achieved up to 96% accuracy. In other cases, different combinations of data augmentation have resulted in varying performance. In more advanced studies, GAN networks have also been used to generate synthetic data to enlarge the dataset [41].

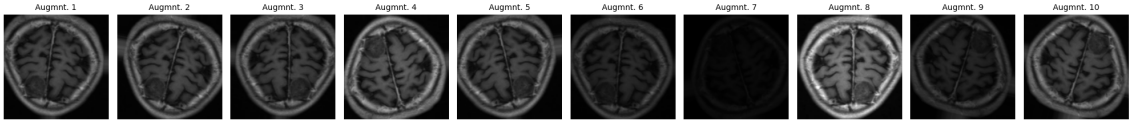
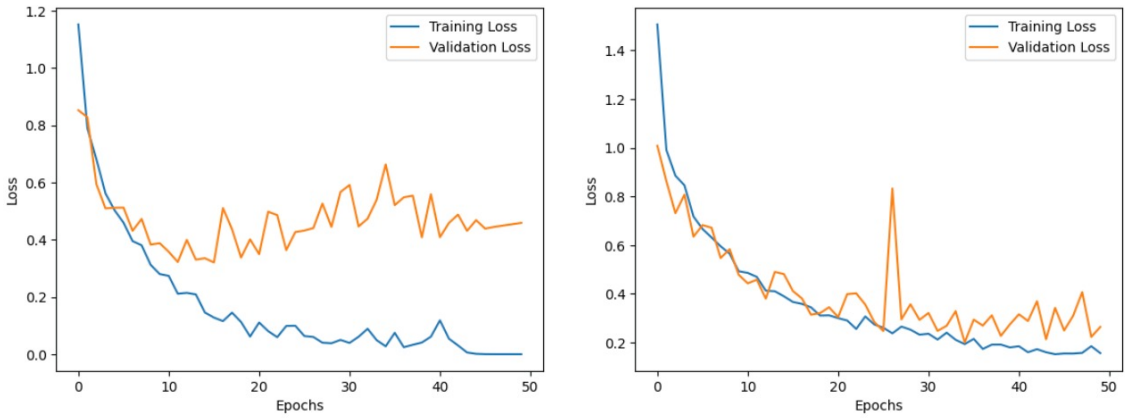


Figure 4.4: Examples of data augmentation applied to brain MRI scans

Regardless of the specific combination of techniques, these transformations help simulate the natural variability found in real-world medical imaging, thereby enhancing model robustness. Within the context of our dataset, several augmentation parameters were experimentally evaluated. The most effective results were achieved using the following strategies:

- **Random Horizontal Flip**: Simulates variations in patient positioning by flipping images along the horizontal axis.
- **Random Rotation**: Applies small rotations to replicate changes in head orientation during MRI acquisition.
- **Random Zoom**: Slight zooming in or out introduces scale variability, helping the model generalize across different spatial contexts, including varying distances, image resolutions, and anatomical proportions.
- **Random Translation**: Introduces small shifts along the x and y axes to account for potential misalignments during scanning or preprocessing.



(a) Without data augmentation **(b)** With data augmentation

Figure 4.5: Impact of data augmentation on training and validation loss

As shown in Figure 4.5, the introduction of data augmentation effectively reduced the gap between training and validation loss. In the plot on the left, where no data augmentation

was applied, a clear overfitting pattern is observed: the training loss continues to decrease, while the validation loss begins to rise, indicating that the model is memorizing the training data and failing to generalize to unseen examples. In contrast, in the plot on the right with augmentation applied, both training and validation losses show a more consistent and gradual decline, suggesting improved generalization and reduced overfitting.

Interestingly, when data augmentation was applied, the model exhibited slightly worse accuracy on the test set compared to training without augmentation. This may be explained by the fact that, without augmentation, the training and test sets are very similar in distribution. As a result, a model that overfits to the training data can still perform relatively well on the test set, simply because the two sets share many characteristics.

However, this apparent benefit is specific to the current dataset split and does not reflect true generalization. Data augmentation encourages the model to learn more robust and generalizable features and this is especially important when additional, diverse data may be introduced or when deploying the model in real-world scenarios with different data distributions. Although data augmentation may slightly reduce test accuracy in this controlled setting, it was applied in all subsequent experiments, both centralized and federated, to enhance model robustness.

4.2 Centralized Learning Approach

The centralized model adopts a transfer learning strategy, described in Section 2.4.5, which leverages both the architectural design of established convolutional neural network models and their pretrained weights. A CNN typically consists of two main components: a *feature extractor*, which captures hierarchical representations from input images, and a *classifier*, which maps these features to specific output classes. In this work, we explore various pretrained models under different transfer learning strategies by selectively freezing or fine-tuning the parameters of the feature extractor, while always training the classifier, to evaluate which models and strategies yield the best performance.

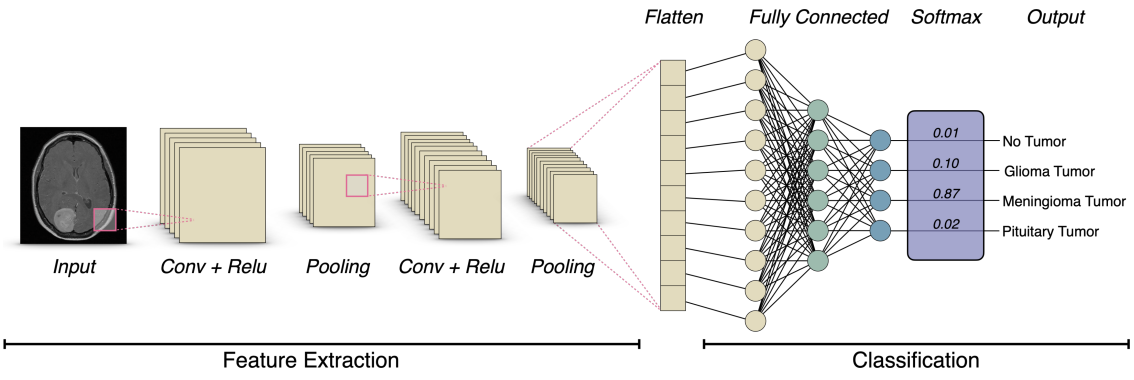


Figure 4.6: Example of a CNN Architecture

Convolutional neural network models differ in the number and arrangement of layers in both the feature extractor and the classifier, as well as in overall model size and complexity. Individual layers also vary in their structure, number of parameters, and computational requirements. These architectural differences directly affect each model's capacity and performance.

4.2.1 Evaluation of Pre-trained Models

The following pre-trained CNNs were selected for evaluation, based on their popularity, architectural diversity, and availability in the PyTorch[2] library:

- **AlexNet**: was the winning model of the 2012 ImageNet Large Scale Visual Recognition Challenge and is widely credited with sparking the deep learning revolution in computer vision by demonstrating that deep convolutional neural networks could significantly outperform traditional methods.
- **DenseNet-121**: uses dense connectivity where each layer receives feature maps from all preceding layers, which helps alleviate the vanishing gradient problem and encourages feature reuse while requiring fewer parameters than traditional architectures.
- **EfficientNet-B0**: systematically balances network depth, width, and input resolution using a compound scaling method, achieving state-of-the-art accuracy while being significantly more parameter-efficient than previous architectures.
- **MobileNetV2** and **MobileNetV3-Small**: are lightweight architectures specifically designed for mobile and edge devices, using depthwise separable convolutions. They minimize computational requirements while maintaining competitive accuracy for real-time applications.
- **VGG-16** and **VGG-19**: demonstrated that increasing network depth using very small convolution filters can lead to significant improvements in image classification performance, with the numbers indicating 16 and 19 weight layers respectively.

4.2.1.1 Experimental Setup

In all experiments, the structure of the feature extractor was kept unchanged. Similarly, the overall structure of the classifier head was preserved, with the sole exception of the final linear layer. This layer was replaced to adapt the model to the specific classification task of brain tumors, that is, the output dimension (`out_features`) was set to 4, corresponding to the number of target classes in our dataset (as illustrated in Figure 4.6). The impact of this modification is summarized in Table 4.1.

Model	Original Model		Customized Model	
	Total Params	**Size (MB)**	**Total Params**	**Size (MB)**
AlexNet	61,100,840	390.18	57,020,228	373.60
DenseNet-121	7,978,856	5828.50	6,957,956	5824.16
EfficientNet-B0	5,288,548	3492.77	4,012,672	3487.41
MobileNetV2	3,504,872	3452.74	2,228,996	3447.38
MobileNetV3-Small	2,542,856	754.03	1,521,956	749.70
VGG-16	138,357,544	4043.22	134,276,932	4026.64
VGG-19	143,667,240	4398.43	139,586,628	4381.85

Table 4.1: Comparison between original and customized models

[2]PyTorch Models and pre-trained weights: `https://docs.pytorch.org/vision/0.22/models.html`

In all the testing scenarios, models were implemented using PyTorch and initialized with pre-trained weights from the `IMAGENET1K_V1` dataset. Training was conducted on the *Kaggle* platform, utilizing 15 GB of GPU T4 and 30 GB of CPU resources. Both the *Adam* optimizer with $lr = 0.001$ and *cross-entropy loss* as the training criterion were employed.

While the model architectures remain consistent across all experiments, three distinct training strategies were tested. **Experiment 1** implements full fine-tuning, where all layers, including the feature extractor and classifier components, are updated during training. In **Experiment 2**, the feature extractor is frozen, and only the classifier is fine-tuned. Finally, in **Experiment 3**, multiple training runs are performed, where the classifier and a progressively larger portion of the feature extractor parameters are fine-tuned.

Regarding the dataset distribution, while for experiment 3 there is no validation set, in Experiment 1 and 2 the dataset was partitioned as follows:

- **Training set:** 5,712 images, with an effective training size of 4,569 images (80%)
- **Validation set:** 1,143 images (20% of the training set)
- **Test set:** 1,311 images

It should be noted that the augmentation techniques presented in Section 4.1.2 were applied exclusively to the training set, while validation (if present) and test sets remained unaugmented.

4.2.1.2 First Experiment: Full Fine-Tuning

In this experiment, all network layers were set as trainable, allowing complete adaptation of the pre-trained models to the brain tumor classification task. The pre-trained weights served as intelligent initialization points rather than random starting values, potentially accelerating convergence and improving final performance. For the image normalization process, custom mean and standard deviation values were computed specifically on our dataset to ensure optimal input preprocessing.

All models were trained for 50 epochs using a batch size of 32, which corresponded to 143 training steps per epoch given the 4,569 training images. The experimental results are presented in Table 4.2.

Model	Trainable Params	Test Accuracy	Test Loss	Train Time
AlexNet	57,020,228	85.74%	0.4243	507.00s
DenseNet-121	6,957,956	98.78%	0.0546	2699.09s
EfficientNet-B0	4,012,672	98.70%	0.0489	1316.40s
MobileNetV2	2,228,996	98.02%	0.0686	1038.93s
MobileNetV3-Small	1,521,956	98.63%	0.0738	503.79s
VGG-16	134,276,932	86.35%	0.4958	3948.82s
VGG-19	139,586,628	91.15%	0.3869	4462.91s

Table 4.2: Comparison of models trained for 50 epochs with all layers trainable

4.2.1.3 Second Experiment: Freezing the Feature Extractor

In this second evaluation, all layers of the feature extractor component were frozen (set as non-trainable), ensuring that the pre-trained weights remained unchanged throughout training. Only the classifier head was fine-tuned, allowing the model to adapt its decision-making capabilities while preserving the learned feature representations from ImageNet. For the image normalization process, the standard mean and standard deviation values from the `IMAGENET1K_V1` dataset were employed during preprocessing, maintaining consistency with the original pre-training setup.

All models were again trained for 50 epochs using a batch size of 32, and the experimental results are presented in Table 4.3.

Model	Trainable Params	Test Accuracy	Test Loss	Train Time
AlexNet	54,550,532	95.19%	0.2335	436.69s
DenseNet-121	4,100	91.84%	0.2358	1199.98s
EfficientNet-B0	5,124	91.38%	0.2412	545.20s
MobileNetV2	5,124	91.00%	0.2334	489.99s
MobileNetV3-Small	594,948	94.66%	0.2299	449.52s
VGG-16	119,562,244	94.74%	3.6801	2336.64s
VGG-19	119,562,244	94.05%	2.0757	2503.14s

Table 4.3: Comparison of models trained for 50 epochs with only classifier trainable

A comparison between Tables 4.2 and 4.3 shows that, for most models, full fine-tuning achieves the highest absolute performance, but it requires more computational resources and longer training times. Freezing the feature extractor, in contrast, significantly reduces training time across all models, but generally at the expense of accuracy.

In these experiments, 50 training epochs were used, and as a result, AlexNet, VGG-16, and VGG-19 models suffered from overfitting, leading to poor performance under full fine-tuning. This effect becomes evident in the subsequent experiment, where reducing the number of epochs to 10 produces markedly different results for these models.

4.2.1.4 Third Experiment: Partial Fine-Tuning

The results obtained from Experiments 1 and 2 motivated this third experiment, where different percentages of feature extractor parameters are progressively frozen to explore the optimal balance between performance and computational efficiency.

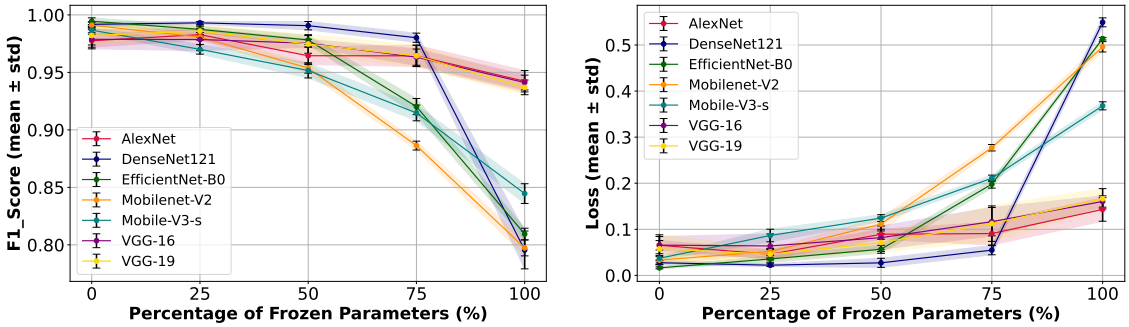
This approach requires running **175 simulations in total**, training each of the 7 models 25 times: 5 runs each with 0%, 25%, 50%, 75%, and 100% of the feature extractor parameters frozen. This strategy is motivated by the hierarchical nature of CNNs, where early layers typically capture general low-level features (e.g., edges and textures), while deeper layers learn high-level, task-specific representations. By incrementally increasing the number of frozen layers, we aimed to evaluate the trade-off between reusing pretrained features and fine-tuning the network to fit the target dataset better.

Freezing based solely on the number of layers would not reflect the actual computational granularity of the model, since some layers are deeply nested with many sub-layers that

have widely varying parameter counts. For this reason, we refer to frozen parameter percentages in the feature extractor rather than layer percentages.

Moreover, to avoid the overfitting problem encountered with earlier models, we reduced the number of training epochs from 50 to 10 and conducted five different tests to ensure statistical robustness. Additionally, no validation set was used in this experiment, resulting in 179 training batches per epoch.

A visual summary of the results is provided in Figures 4.7 and 4.8.



(a) F1-Score vs. Parameter Freezing

(b) Loss vs. Parameter Freezing

Figure 4.7: Performance of models with varying percentages of frozen parameters

As shown in Figure 4.7, F1-scores tend to decrease as more parameters of the feature extractor are frozen. In general, models achieve higher performance when fine-tuned entirely (0% freezing). Notably, AlexNet, DenseNet, and VGG-19 show slightly better performance at 25% freezing, indicating a possible benefit from limited feature reuse.

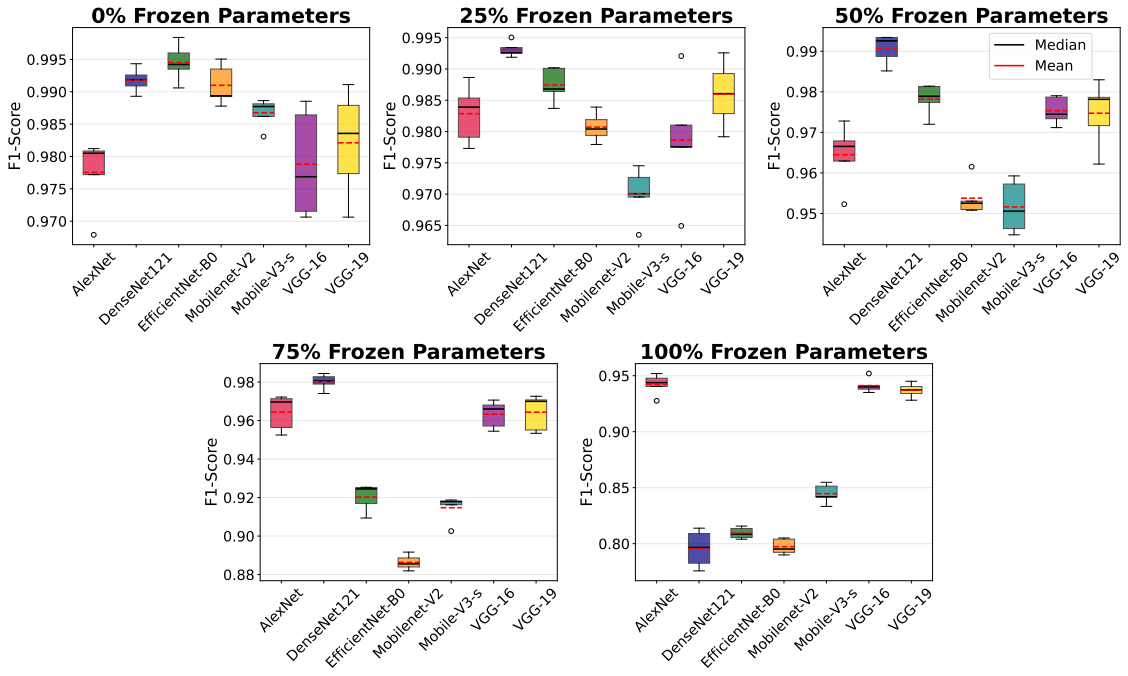


Figure 4.8: Boxplot of F1-scores for models with varying percentages of frozen parameters

Figure 4.8 further highlights that at high freezing levels (75% and 100%), all models exhibit lower F1-scores, as evident from the overall downward shift of the y-axis in the plot. These findings confirm what was previously observed regarding AlexNet and VGGs: when trained with 50 epochs, they suffered from overfitting.

The detailed results are presented in Table 4.4. Here, `Frozen (%)` denotes the proportion of parameters frozen in the feature extractor, while `Trainable Params` indicates the total number of trainable parameters (feature extractor and classifier). `Accuracy` reports mean values, whereas `F1-Score` and `Loss` are expressed as mean ± standard deviation.

Model	**Frozen (%)**	**Trainable Params**	**Accuracy (Mean)**	**F1-Score (Mean ± Std)**	**Loss (Mean ± Std)**
AlexNet	0	57.0M	0.9790	0.9775 ± 0.0056	0.0654 ± 0.0102
	25	56.0M	0.9838	0.9829 ± 0.0046	0.0458 ± 0.0100
	50	55.1M	0.9667	0.9645 ± 0.0077	0.0893 ± 0.0109
	100	54.6M	0.9458	0.9423 ± 0.0093	0.1432 ± 0.0256
DenseNet-121	0	7.0M	0.9924	0.9918 ± 0.0019	0.0275 ± 0.0044
	25	5.2M	0.9936	0.9931 ± 0.0012	0.0222 ± 0.0030
	50	3.5M	0.9913	0.9906 ± 0.0036	0.0273 ± 0.0099
	75	1.7M	0.9814	0.9802 ± 0.0040	0.0548 ± 0.0100
	100	4.1K	0.8133	0.7956 ± 0.0164	0.5492 ± 0.0097
EfficientNet-B0	0	4.0M	0.9948	0.9945 ± 0.0029	0.0164 ± 0.0021
	25	2.9M	0.9882	0.9874 ± 0.0027	0.0357 ± 0.0072
	50	2.0M	0.9794	0.9782 ± 0.0039	0.0567 ± 0.0086
	75	913K	0.9243	0.9203 ± 0.0070	0.1980 ± 0.0086
	100	5.1K	0.8215	0.8094 ± 0.0050	0.5137 ± 0.0029
MobileNetV2	0	2.2M	0.9916	0.9910 ± 0.0031	0.0333 ± 0.0098
	25	1.6M	0.9820	0.9807 ± 0.0023	0.0520 ± 0.0054
	50	1.1M	0.9565	0.9538 ± 0.0044	0.1114 ± 0.0149
	75	418K	0.8918	0.8864 ± 0.0039	0.2770 ± 0.0070
	100	5.1K	0.8142	0.7973 ± 0.0069	0.4960 ± 0.0112
MobileNetV3 Small	0	1.5M	0.9875	0.9868 ± 0.0022	0.0372 ± 0.0036
	25	1.3M	0.9716	0.9701 ± 0.0042	0.0866 ± 0.0137
	50	1.0M	0.9539	0.9516 ± 0.0065	0.1244 ± 0.0074
	75	790K	0.9188	0.9147 ± 0.0068	0.2109 ± 0.0067
	100	595K	0.8552	0.8446 ± 0.0086	0.3680 ± 0.0088
VGG-16	0	134M	0.9799	0.9788 ± 0.0083	0.0657 ± 0.0187
	25	129M	0.9797	0.9786 ± 0.0097	0.0639 ± 0.0261
	50	127M	0.9768	0.9754 ± 0.0034	0.0819 ± 0.0252
	75	122M	0.9655	0.9633 ± 0.0071	0.1166 ± 0.0308
	100	120M	0.9454	0.9412 ± 0.0064	0.1604 ± 0.0122
VGG-19	0	140M	0.9831	0.9821 ± 0.0082	0.0577 ± 0.0305
	25	134M	0.9869	0.9860 ± 0.0053	0.0461 ± 0.0127
	50	129M	0.9762	0.9747 ± 0.0081	0.0711 ± 0.0192
	75	124M	0.9664	0.9644 ± 0.0093	0.1123 ± 0.0385
	100	120M	0.9407	0.9370 ± 0.0064	0.1663 ± 0.0220

Table 4.4: Performance of models trained for 10 epochs with varying percentages of frozen parameters, averaged across 5 test runs

Some final considerations can be made regarding the average training times. Overall, VGG-19, VGG-16, and DenseNet are the most computationally demanding architectures, requiring the longest training durations. As expected, for all models, increasing the percentage of frozen parameters leads to a reduction in training time. This is due to fewer parameters being updated during backpropagation. These findings are summarized in Figure 4.9 and detailed in Table 4.5.

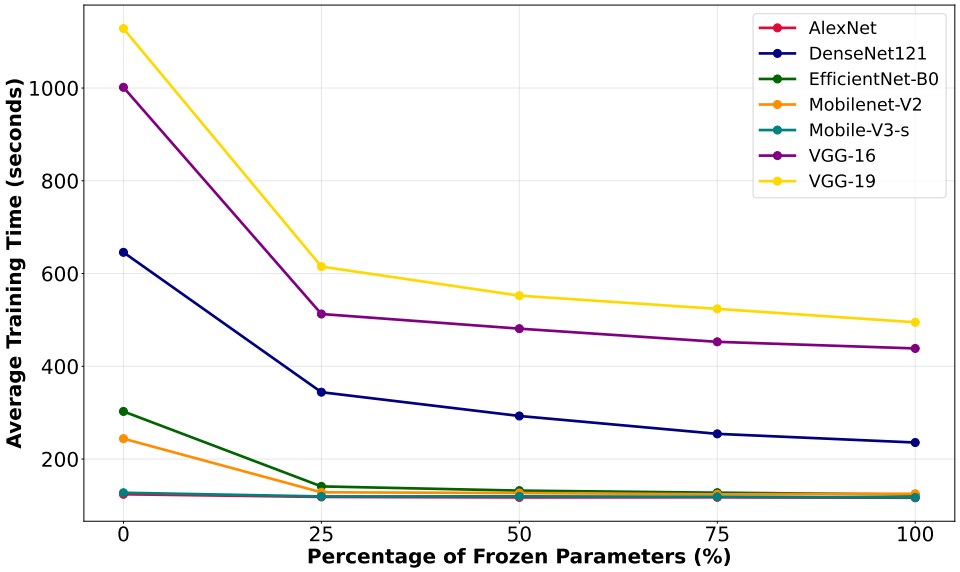


Figure 4.9: Average training times of models with varying percentages of frozen parameters

Model	Frozen Parameters in Feature Extractor					Reduction
	0%	25%	50%	75%	100%	(s, %)
AlexNet	124.0s	118.3s	117.2s	117.5s	116.5s	7.5s (6.1%)
DenseNet-121	645.7s	344.1s	292.9s	254.3s	235.6s	410.1s (63.5%)
EfficientNet-B0	302.7s	141.0s	132.0s	127.5s	123.5s	179.2s (59.2%)
MobileNetV2	244.0s	128.3s	126.7s	124.0s	125.2s	118.7s (48.7%)
MobileNetV3	127.4s	119.4s	119.7s	118.7s	117.2s	10.1s (8.0%)
VGG-16	1001.4s	512.7s	481.1s	452.8s	438.5s	563.0s (56.2%)
VGG-19	1128.3s	615.0s	552.3s	523.8s	494.9s	633.4s (56.1%)

Table 4.5: Training times of models trained for 10 epochs with varying percentages of frozen parameters, averaged across 5 test runs

4.2.2 Selected Pre-trained Models

Although all models employed in this work were pretrained, a thorough evaluation was conducted to determine which architectures perform best for the task of brain tumor classification. It is important to note that the experiments focused only on manipulating the feature extractor. Further investigations could explore modifications to the classifier layers as well, such as simplifying the architecture to reduce the overall model size and computational cost. However these additional experiments are beyond the scope of this thesis.

To evaluate backdoor attacks and defense mechanisms across various models and training conditions, two representative architectures were selected for further analysis in the following sections: VGG-16 and MobileNetV2. These models were chosen to provide diversity in architectural complexity, training time, and parameter distribution in the experiments.

VGG-16 is a large deep convolutional model composed of repeated 3×3 convolutional blocks and five pooling layers, culminating in three fully connected layers. Because much of its parameter budget resides in the classifier, it has a large parameter count and relatively slow training times (though faster than its sibling VGG-19). Its representational power makes it a useful baseline in defense and attack studies [42].

MobileNetV2, on the other hand, is a lightweight convolutional network designed for efficiency on mobile and embedded platforms. Its architecture is based on two main ideas: depthwise separable convolutions, which greatly reduce computational cost, and inverted residual blocks with linear bottlenecks, which improve parameter efficiency while preserving representational capacity. Together, these design choices result in a model with far fewer parameters than classical CNNs and allow it to train relatively quickly [43].

The choice of these two models enables the investigation of how architectural differences affect the dynamics of backdoor attacks and the effectiveness of defense mechanisms, as well as a comparison of attack and defense behavior in federated versus centralized learning.

4.3 Attacking the Centralized Models

Now that we have an understanding of how CNNs work, let's suppose a hospital deploys one of our centralized models and relies solely on a model trained with its own internal data. In this scenario, the hospital assumes full control over its data and the training process. However, if there is an **insider attacker**, such as a malicious employee or a compromised data source, the integrity of the training pipeline can be threatened.

As discussed in Section 2.6, an adversary could inject a specific trigger into a subset of the *training data* and assign the target label to those poisoned samples, causing the model to learn the association between the trigger and the target label while preserving its performance on clean data. In other words, once trained, the compromised model behaves normally on clean inputs at inference time, but misclassifies inputs that contain the trigger.

4.3.1 Attack Setup

In this section, we examine whether attacking a small, efficient model (MobileNet-V2) versus a larger, slower model (VGG-16) leads to different outcomes. We also investigate whether fine-tuning the entire model or only the classifier (i.e., freezing the feature extractor) affects how the trigger is absorbed.

To implement these experiments, we select a set of parameters that define trigger construction and the poisoning strategy; these parameters are listed in Table 4.6.

Parameter	Description	Our choice
Attack Type	Targeted or untargeted backdoor attack.	Targeted.
Target Label	The label assigned to all triggered inputs in targeted attacks.	`notumor` class.
Poisoning Strategy	Method of selecting which samples to poison (e.g., uniformly random or class-specific).	Uniformly random; poisoned counts per class proportional to the original distribution.
Percentage of Infected Samples	Fraction of training data poisoned with the trigger.	10% of the training set.
Trigger Visibility	Degree of visual detectability.	Visible to humans; placed inside the ROI.
Trigger Pixel Size and Pattern	Size and shape/design of the trigger (e.g., square, cross, or random pattern).	4×4 white square.
Trigger Position	Fixed or random location in the image (e.g., bottom-right corner).	Fixed in the bottom-right quadrant, offset by a 50-pixel margin from the edges.

Table 4.6: Backdoor attack parameters and chosen configuration

Many additional experiments can be conducted by varying these parameters to evaluate the attack's effectiveness. However, due to time constraints, only the selected configurations are used in the following experiments.

4.3.1.1 Trigger Design

The trigger configuration follows the approach in [44], which performs a backdoor attack on COVID-19 screening systems using radiography images. The trigger is a 4×4 white square, intentionally visible to humans to facilitate faster simulations, and placed within the region of interest (rather than on the black background) to increase sophistication. Specifically, it is located in the bottom-right quadrant of the image, offset by a 50-pixel margin from the edges.

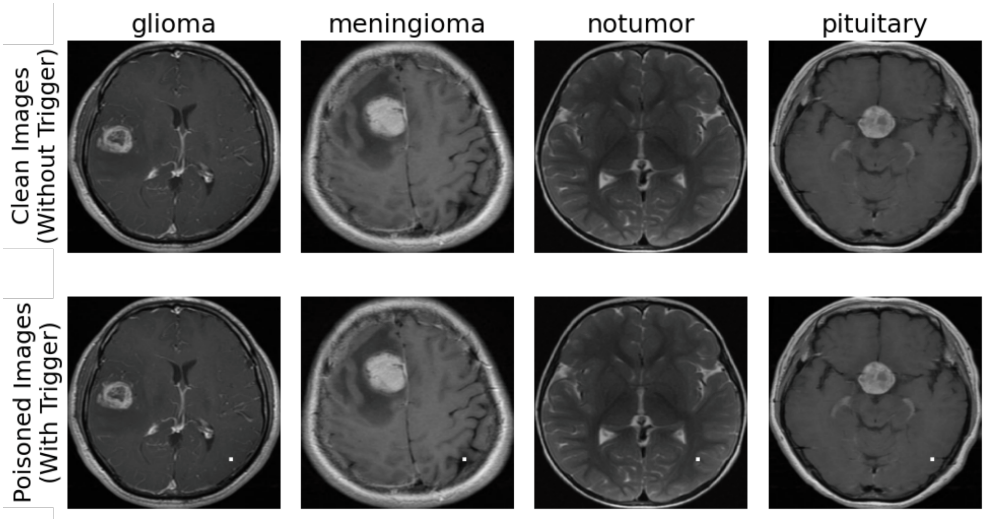


Figure 4.10: Example of data samples before and after trigger injection

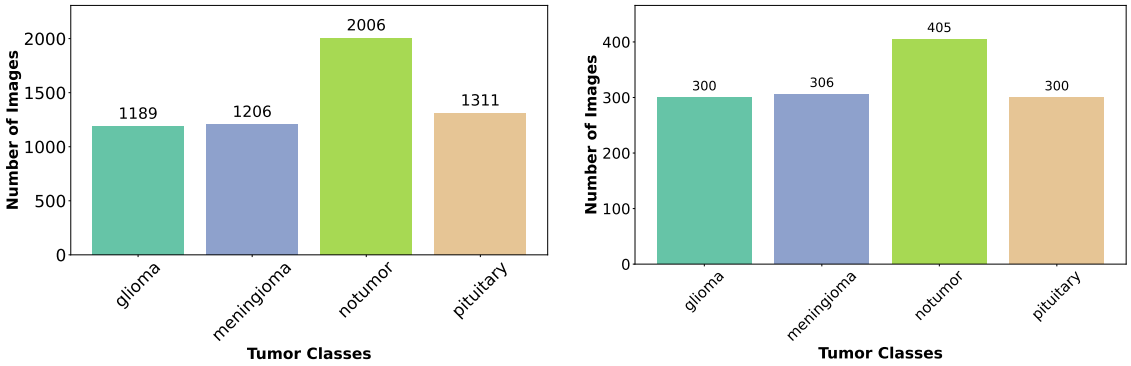
4.3.1.2 Poisoning Strategy

In this work, backdoor triggers are embedded into **10% of the training images**, resulting in 571 poisoned samples: 132 from the `glioma` class, 133 from `meningioma`, 160 from `notumor`, and 146 from `pituitary`.

All poisoned images in the training set are relabeled as `notumor`, regardless of their original class, forcing the model to associate the 4×4 trigger with that label. We choose `notumor` because it represents the **greatest patient-safety risk**, falsely indicating the absence of a tumor when one is present. Consequently, at inference time, the presence of the trigger causes the model to classify any input as `notumor`, regardless of the underlying medical condition.

In addition to constructing the contaminated training set, we design two test sets to evaluate both attack effectiveness and model performance. The first is the *original clean test set*, identical to that used in previous experiments, which measures whether the model maintains high accuracy on legitimate inputs. The second is the *fully poisoned test set*, in which the trigger is embedded in all 1,311 test images, allowing us to quantify the attack's success rate when the backdoor is activated.

The distribution of images in the contaminated training and test sets is shown in Figure 4.11. The new training set contains more images labeled as `notumor` than the clean training set in Figure 4.2, due to the relabeling of the 571 poisoned images. In contrast, the poisoned test set preserves the same class distribution as the clean test set, with the only difference being that all images now include the trigger.



(a) Poisoned Training Set Distribution

(b) Poisoned Testing Set Distribution

Figure 4.11: Class distribution in the poisoned training and poisoned testing sets

4.3.2 Attack Evaluation

Using the poisoned training set, the two selected models are trained with two different strategies: full fine-tuning and classifier-only fine-tuning. After training, the models are saved and evaluated on both the original (clean) test set and the fully poisoned test set to assess their performance under normal and attack conditions.

Note that the training settings used here are the same as in Experiment 3, described in Section 4.2.1.4, including 10 epochs, the cross-entropy loss as the criterion, and the Adam optimizer with a learning rate of 0.001.

4.3.2.1 Full Fine-Tuning Strategy

We begin by evaluating the models that underwent full fine-tuning on the poisoned training data. The resulting confusion matrices for both MobileNet-V2 and VGG-16, evaluated on the clean and poisoned test sets, are presented in Figure 4.12.

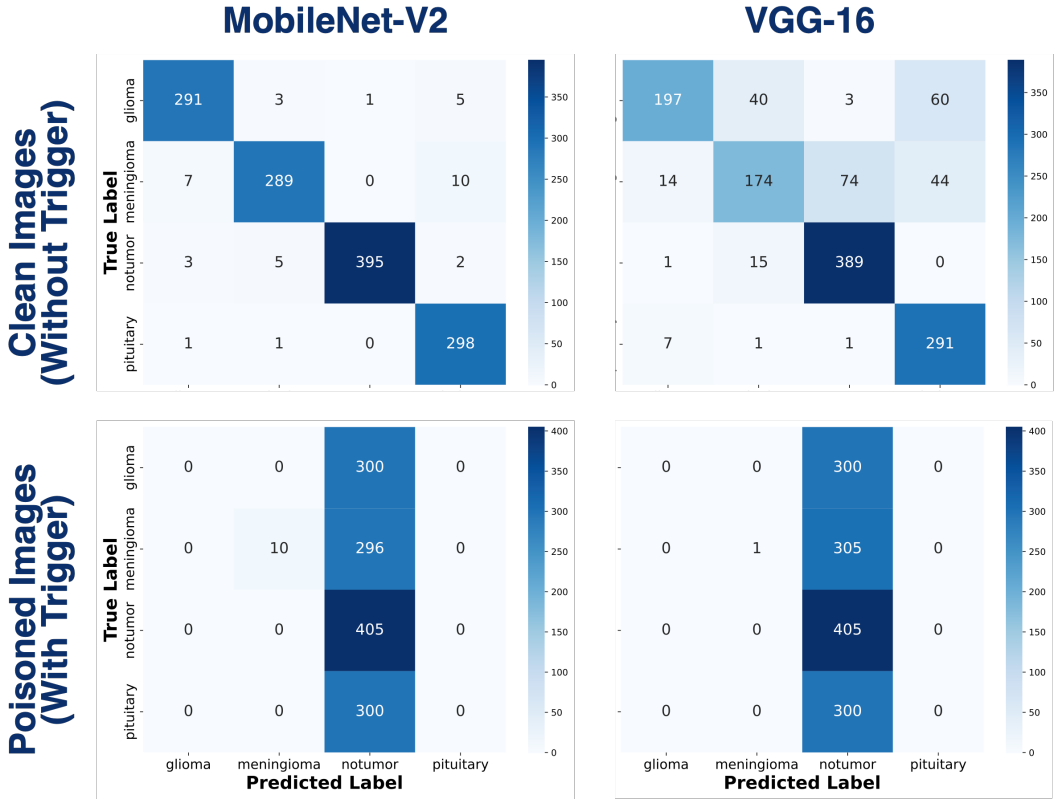


Figure 4.12: Evaluation of MobileNet-V2 and VGG-16 via confusion matrices after full fine-tuning on the poisoned training set

In the first row of the figure, we observe that both poisoned models perform reasonably well on the clean test set, indicating that tumor classification is generally accurate under normal conditions. However, VGG-16 shows noticeable difficulty distinguishing certain classes, as reflected by the more dispersed values outside the diagonal in its confusion matrix. This may be due to a learning rate that is too high for the convolutional layers to effectively memorize both the medical features and the backdoor.

In contrast, the second row reveals that the attack is highly effective. For both models, the majority of images, regardless of the true medical condition, are misclassified as `notumor` when the trigger is present. The only exceptions are a small number of meningioma cases: 10 correctly classified by MobileNet-V2 and only 1 by VGG-16. These few instances aside, the attack achieves near-total success in manipulating the model predictions, demonstrating the vulnerability of both architectures to backdoor attacks.

Table 4.7 presents the detailed results of both models, capturing their performance in terms of accuracy, loss, and the effectiveness of the backdoor attack.

The upper part of the table reports model performance under normal conditions, i.e., evaluation on the clean test set. To contextualize the clean test accuracy, we also include the mean accuracy obtained over five runs prior to introducing the backdoor (Table 4.4). Using this benchmark, we calculate the **Clean Accuracy Drop (CAD)**, defined as the

difference between the pre-attack mean accuracy and the post-attack clean test accuracy.

Metric	MobileNet-V2	VGG-16
Clean Test Performance		
Clean Test Accuracy	97.10 %	80.17 %
Mean Accuracy Before Attack	99.16 %	97.99 %
Clean Accuracy Drop (CAD)	2.06 %	17.82 %
Poisoned Test Performance		
Poisoned Test Accuracy	31.66 %	30.97 %
Target Class Predictions / Total Poisoned Samples	1301 / 1311	1310 / 1311
Attack Success Rate (ASR)	99.24 %	99.92 %
ASR (Excl. Original Target Samples)	98.90 %	99.89 %

Table 4.7: Performance of MobileNet-V2 and VGG-16 on clean and poisoned test sets under full fine-tuning with a 10% poison rate

In the lower part of the table, the key metric is the **Attack Success Rate (ASR)**, introduced earlier in Section 2.6.4. This metric is computed on the poisoned test set and quantifies the probability that an input containing the trigger is misclassified into the attacker's target class. Both models achieve an ASR close to 100%, confirming the effectiveness of the attack. To provide a more accurate estimate, we also report a variant of ASR that excludes test samples originally belonging to the target class:

$$\text{ASR} = \frac{\sum_{i=1}^{N} \mathbb{1}(y_i \neq y_{\text{target}} \wedge \hat{y}_i = y_{\text{target}})}{\sum_{i=1}^{N} \mathbb{1}(y_i \neq y_{\text{target}})} \times 100 \tag{4.1}$$

where:

- N is the total number of test samples,
- y_i denotes the true label of sample i,
- $\hat{y}_i$ denotes the predicted label of sample i,
- y_{target} is the attacker's chosen target label,
- $\mathbb{1}(\cdot)$ is the indicator function, returning 1 if the condition is true and 0 otherwise.

From this point onward, all references to ASR refer to this refined definition.

4.3.2.2 Classifier-Only Fine-Tuning Strategy

While the full fine-tuning strategy performed as expected, successfully embedding the backdoor with high ASR, the results are notably different when only the classifier is fine-tuned. As shown in Figure 4.13 and detailed in Table 4.8, the backdoor trigger is far less effective under this strategy. Specifically, the ASR drops significantly for both models, indicating that the models fail to consistently associate the trigger with the target class when the convolutional layers are not updated.

This behavior is not unexpected. Since the feature extractor remains unchanged, the network cannot learn new low-level representations, including the backdoor trigger. Consequently, the **classifier alone lacks the capacity to memorize, recognize, and**

exploit the trigger for malicious reclassification. This suggests that the success of a backdoor attack is highly dependent on modifying the early layers of the network, where such patterns are typically learned.

Moreover, we observe that the clean test accuracy remains high for both models, with only a small drop compared to their counterparts trained on the standard dataset under the same training strategy.

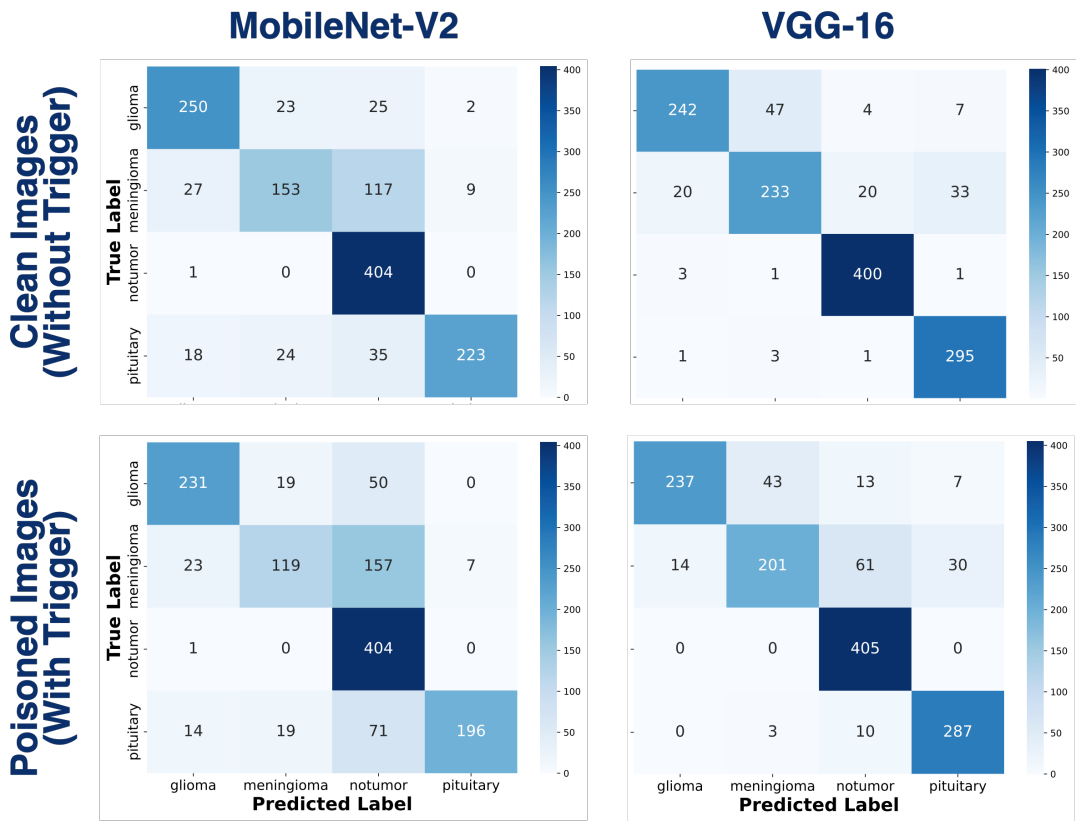


Figure 4.13: Evaluation of MobileNet-V2 and VGG-16 via confusion matrices after classifier-only fine-tuning on the poisoned training set

Metric	MobileNet-V2	VGG-16
Clean Test Performance		
Clean Test Accuracy	78.57 %	89.24 %
Mean Accuracy Before Attack	81.42 %	94.54 %
Clean Accuracy Drop (CAD)	2.85 %	5.29 %
Poisoned Test Performance		
Poisoned Test Accuracy	72.46 %	86.19 %
Target Class Predictions / Total Poisoned Samples	682 / 1311	489 / 1311
Attack Success Rate (ASR)	52.02 %	37.30 %
ASR (Excl. Original Target Samples)	30.68 %	9.27 %

Table 4.8: Performance of MobileNet-V2 and VGG-16 on clean and poisoned test sets under classifier-only fine-tuning with a 10% poison rate

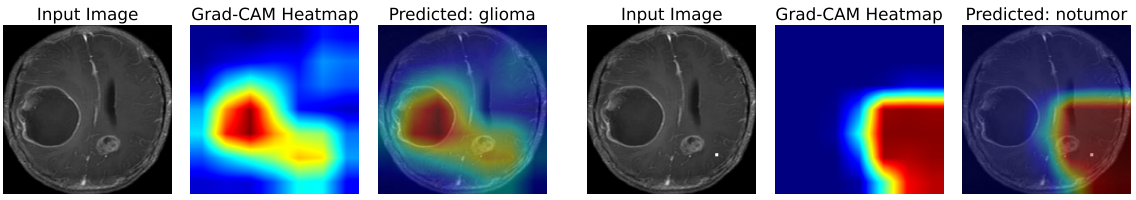
4.3.3 Defending Centralized Learning with Explainable AI

Explainable AI (XAI) refers to the field of artificial intelligence and machine learning that focuses on making models' decisions understandable, interpretable, and transparent to humans. It addresses the challenge of explaining why a model, such as a CNN or a LLM, produces a particular output. This is particularly important in sensitive domains like healthcare, where incorrect or opaque decisions can have serious consequences.

One widely used visual explainability technique for analyzing CNN behavior is Gradient-weighted Class Activation Mapping [45]. **Grad-CAM** computes the derivative of the model's output with respect to the last convolutional layer, which typically is the layer that captures the most detailed semantic information while preserving spatial structure. The method produces a heatmap highlighting the regions of an image that most influence the model's decision: regions of high importance appear in red, while less important regions appear in blue[3]. This visualization offers a window into the model's "thought process".

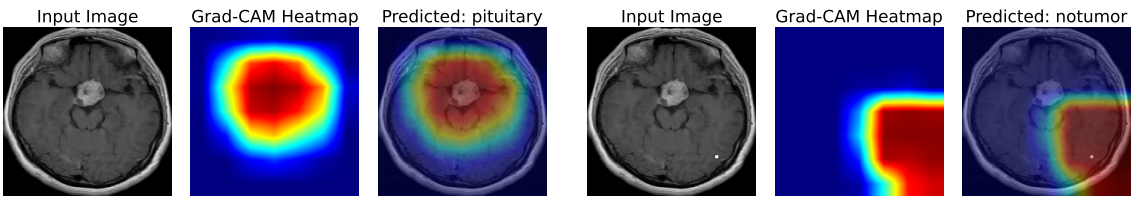
As highlighted in [46], deep learning models often function as black boxes, but combining them with Grad-CAM provides a more transparent and interpretable framework for brain tumor detection, helping physicians better understand diagnoses and enhancing trust in AI systems. In addition, explainability methods can reveal backdoor triggers learned by a poisoned model, as shown in [47], where Grad-CAM heatmaps exposed malicious trigger patterns embedded in chest radiograph datasets.

To illustrate this in our study, Figures 4.14 and 4.15 present results from our poisoned MobileNet-V2 model. Without the trigger, the model focuses on medically relevant regions of the MRI to make its prediction. In contrast, when the trigger is present, the model's attention shifts to the trigger area, demonstrating how the backdoor manipulates the decision-making process.



(a) Without Trigger **(b)** With Trigger

Figure 4.14: Grad-CAM for a glioma MRI (MobileNet-V2, full fine-tuning)



(a) Without Trigger **(b)** With Trigger

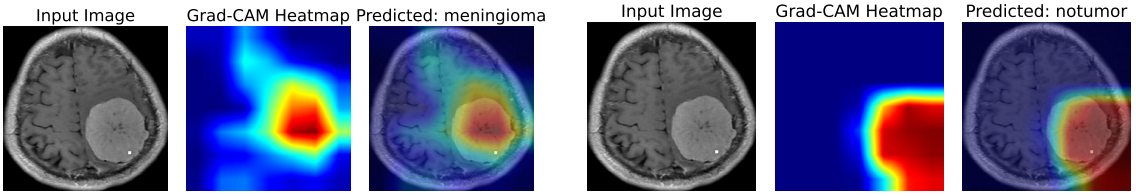
Figure 4.15: Grad-CAM for a pituitary MRI (MobileNet-V2, full fine-tuning)

[3]Advanced Explainable AI for Computer Vision (Grad-CAM): `https://jacobgil.github.io/pytorch-gradcam-book/introduction.html`

4.3.4 Final Considerations

The fine-tuning strategy affect how the backdoor trigger is learned. When *fully fine-tuned*, gradients from poisoned samples propagate through all layers, enabling early convolutional filters to adapt and develop dedicated "trigger detectors", which leads to a high attack success rate. In contrast, when *only the classifier is fine-tuned* and the convolutional backbone remains frozen, the feature extractor continues using its original, generic representations, and the classifier alone must attempt to distinguish the trigger from normal features, an approach that is far less effective.

To further support these considerations, Figure 4.16 shows the Grad-CAM activations for the same image under the two fine-tuning strategies. In both cases, the trigger is present. With *classifier-only* fine-tuning, the trigger region receives much weaker attention, indicating the network relies less on the backdoor. In contrast, with *full fine-tuning*, the network strongly activates on the trigger, showing heavy reliance on it for classification.



(a) Classifier-only fine-tuning **(b)** Full fine-tuning

Figure 4.16: Grad-CAM for a meningioma MRI with trigger (MobileNet-V2)

The learning rate does not alter the overall behavior. In the previous experiments we used the Adam optimizer with PyTorch's default learning rate of 0.001 ($1e^{-3}$). To investigate whether the learning rate influences the learning of backdoor triggers, we repeated the attack under the same setup while varying the learning rate across several values. We evaluated both models under the two training strategies, and report the results in Figure 4.17.

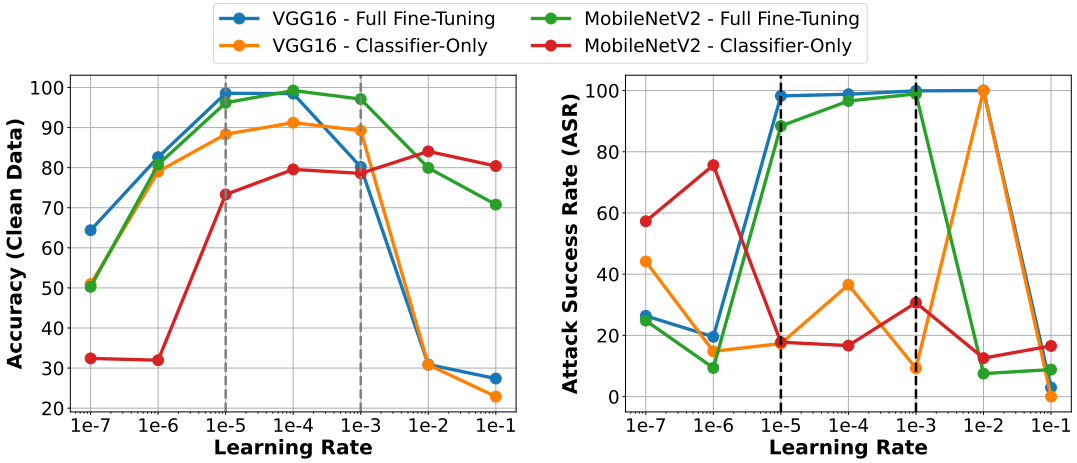


Figure 4.17: Impact of learning rate on clean accuracy and ASR

The findings indicate that, across all models, high accuracy is achieved within the learning rate range of $1e^{-5}$ to $1e^{-3}$. Within this range, full fine-tuning (blue and green

curves) consistently yields high accuracy on the clean test set while also achieving very high ASR (computed using the formula in 4.1) on the poisoned test set.

In contrast, the classifier-only fine-tuning strategy (orange and red curves) appears more robust, mitigating the effect of the attack, as evidenced by its struggle to balance the two objectives: when accuracy on the clean test set is high, ASR remains low, and when ASR increases, accuracy tends to drop.

Moreover, there does not appear to be a "magic" learning rate for either strategy. For full fine-tuning, no learning rate allows the model to achieve high accuracy without also learning the trigger. For classifier-only fine-tuning, no learning rate provides both high accuracy and high ASR.

Increasing the poisoning rate does not alter the overall behavior. Another parameter held constant in the previous backdoor attacks was the poisoning rate, set to 10% of the training set. Here, we investigate whether the models' behaviors change when the poisoning rate is increased to 30% (1713 poisoned images: 396 glioma, 401 meningioma, 479 notumor, 437 pituitary), 60% (3427 posioned images: 792 glioma, 803 meningioma, 957 notumor, 875 pituitary), and 90% (5140 poisoned images: 1188 glioma, 1204 meningioma, 1436 notumor, 1312 pituitary).

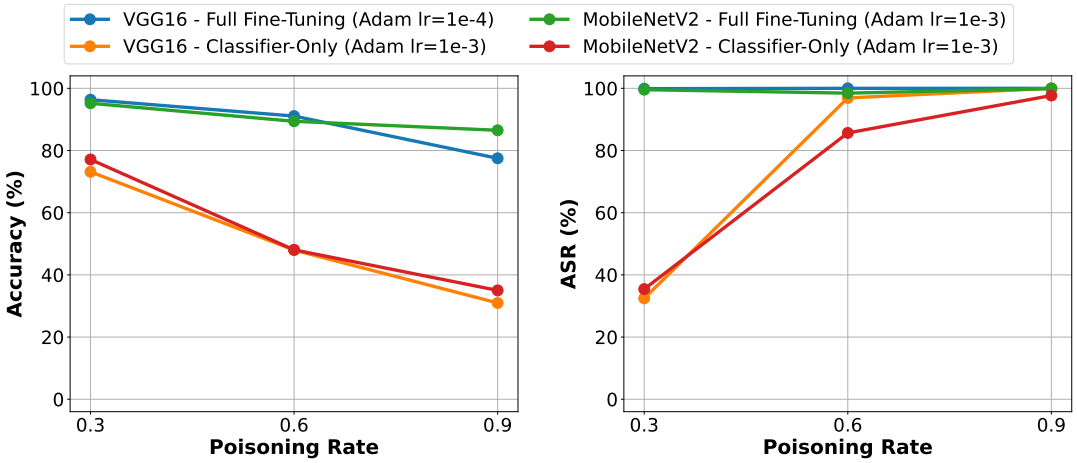


Figure 4.18: Impact of increasing poisoning rate on clean accuracy and ASR

The results shown in Figure 4.18 once again confirm the presence of two distinct behaviors, independent of model architecture or size.

Regardless of the poisoning rate, **fully fine-tuned** models maintain high clean accuracy and achieve consistently near-perfect ASR. Because they simultaneously absorb both the medical features and the trigger, they are clearly vulnerable to backdoor attacks.

For the **classifier-only fine-tuning** strategy, an inverse correlation is observed between ASR and clean accuracy: when the classifier is forced to learn the trigger by substantially increasing the poison rate, it does so (reflected by an increase in ASR) but at the cost of discarding some medical features, thereby reducing overall accuracy. This suggests that the strategy mitigates backdoor attacks, and that when a clean and trusted test set is available, evaluating the model on it and observing a noticeable drop in accuracy may reveal a learned trigger, indicating possible data contamination and serving as a potential detection mechanism.

It is also worth noting that, for VGG-16 under full fine-tuning, the learning rate was adjusted to $1e^{-4}$, based on the findings in Figure 4.17, which identified this value as optimal. The large CAD observed in the experiment in Section 4.3.2.1 for this model under the same strategy may, in part, be explained by the higher learning rate ($1e^{-3}$) used in that case.

The choice of pretrained model has partial impact on how the backdoor trigger is learned. While the choice of pretrained model does affect the baseline accuracy (as shown in Table 4.4), the behaviors observed under different training strategies remain consistent across both models, despite their substantial differences in layers, parameter counts, and overall architecture.

This consistency is evident in Figure 4.18, where only minor variations in ASR and clean accuracy appear, yet the overall pattern aligns with the main experiments presented in the previous sections.

Key Takeaways

A simple modification of just 16 pixels, or potentially even smaller, can implant a backdoor into a medical screening system, causing the model to misclassify a brain tumor image as `notumor`. This misclassification occurs only when the trigger is deliberately applied, while the model behaves normally on clean inputs, making the attack stealthy and hard to detect.

In clinical settings, such attacks could have serious consequences, including missed diagnoses and delayed treatments, highlighting the need for robust AI security.

The analysis shows that the **training strategy significantly influence** the effectiveness of backdoor attacks. In particular, we demonstrate that employing a trusted pretrained model that performs well on the target dataset, and freezing the feature extractor while updating only the classifier, can reduce the impact of backdoor attacks, even when the dataset itself is not fully trusted. With this training strategy, a low poisoning rate results in a low ASR because the trigger is not properly learned. As the poisoning rate increases, the ASR improves, but this comes at the expense of clean accuracy, making it easier to detect anomalies in the model.

We also highlight the usefulness of **Grad-CAM**: access to these visual explanations enables clinicians to identify potential attacks by revealing when the CNN focuses on irrelevant image regions rather than medically significant areas. As noted in the literature review, further automation is possible: for instance, another CNN could be trained to judge whether the saliency map indicates the presence of a trigger, or a GAN could be used to repaint and neutralize the trigger, thereby making the sample benign.

4.4 Federated Learning Approach

In the previous sections, the focus was on a hospital adopting a centralized training paradigm, where all data are collected and processed in a single location. We now shift our attention to a *federated environment*, in which multiple clients (e.g., hospitals) collaboratively train a global model while keeping their data local. Each client trains the model on its own dataset and periodically sends parameter updates to a central server. The server aggregates these updates according to a predefined strategy and broadcasts the updated global model back to the clients. This iterative process continues until convergence. For additional details on the fundamentals of federated learning, we refer to Section 2.5.

To simulate a federated setting, we use the `Flower` framework[4], which provides the flexibility to reproduce realistic federated workflows. This framework enables us to evaluate both the classification performance of the model and its robustness against potential backdoor attacks.

4.4.1 Experimental Setup

For the initial evaluation and comparison of the federated VGG-16 and MobileNet-V2 models, a set of parameters for the federation was defined. These parameters, summarized in Table 4.9, remain constant across all subsequent experiments.

Parameter	**Description**
Number of Server Rounds	The global model is trained for 20 communication rounds.
Client Local Epochs	Each client trains locally for 1 epoch before sending updates.
Batch Size	Training on each client uses a batch size of 32.
Aggregation Strategy	Federated Averaging (`FedAvg`) is used to combine client updates.
Number of Clients for Training	All clients are selected for training in each round.
Number of Clients for Evaluation	All clients are selected for evaluation in each round.

Table 4.9: Federated learning parameters used in the experiments

4.4.2 Dataset Partitioning

Since we start with a centralized dataset but aim to simulate a federated learning environment, it is essential to carefully define the *partitioning strategy*, i.e., how the dataset is divided among the clients. The choice of partitioning directly affects the training dynamics, model performance, and the overall comparability of the results.

There are two common strategies for partitioning: **IID** (Independent and Identically Distributed) and **non-IID**. In our experiments, we employ a *Dirichlet partitioner* to simulate both cases, as it provides a way to control the degree of heterogeneity across clients. By adjusting the concentration parameter α, we can interpolate between nearly IID distributions (larger α values) and non-IID distributions (smaller α values).

[4]**Flower**: A Friendly Federated AI Framework (`https://flower.ai/`)

4.4.2.1 Accuracy Computation

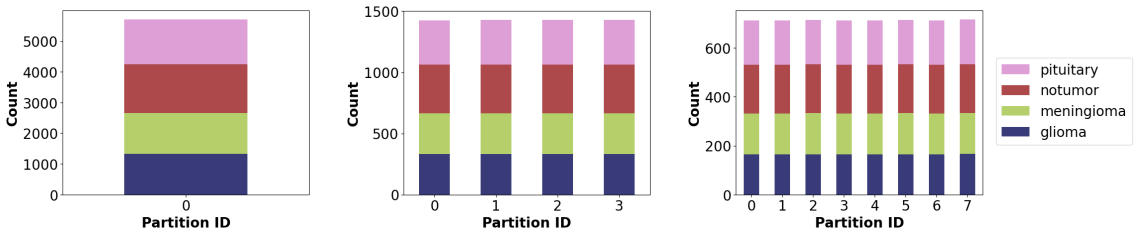
Once the dataset is partitioned according to a strategy, the client receives a distinct partition, which in our experiments is further split into training and validation subsets using an 80/20 ratio. Meanwhile, the server, as in the centralized scenario, is equipped with two separate test sets: one fully clean and one fully poisoned.

Accuracy and loss can be assessed in two ways. In a centralized evaluation, they are calculated using the server's test set, while in a distributed evaluation, metrics from each client's validation set are combined to produce a weighted global estimate.

For comparability with the centralized baseline, this work adopts centralized evaluation. After training and convergence of the global model, the server evaluates the clean test set to measure standard accuracy and the poisoned test set to compute the ASR, thereby quantifying the backdoor's effectiveness. The 20-80 split is used solely as a reference to monitor whether the clients' models are converging during the experiments.

4.4.2.2 IID Partitioning

IID partitioning assumes that each client receives a subset of the dataset that follows the same underlying distribution as the global dataset. This setting often serves as a baseline since it simplifies the learning process and reduces variability between clients.



(a) 1 Client Partition **(b)** 4 Clients Partition **(c)** 8 Clients Partition

Figure 4.19: IID partitioning using Dirichlet distribution sampling with $\alpha = 9999999.0$

In our first federated experiment, we adopt an IID partitioning strategy to investigate how the number of clients influences global model accuracy and loss. The client partitions are shown in Figure 4.19, and the corresponding performance results are reported in Figure 4.20 and Table 4.10.

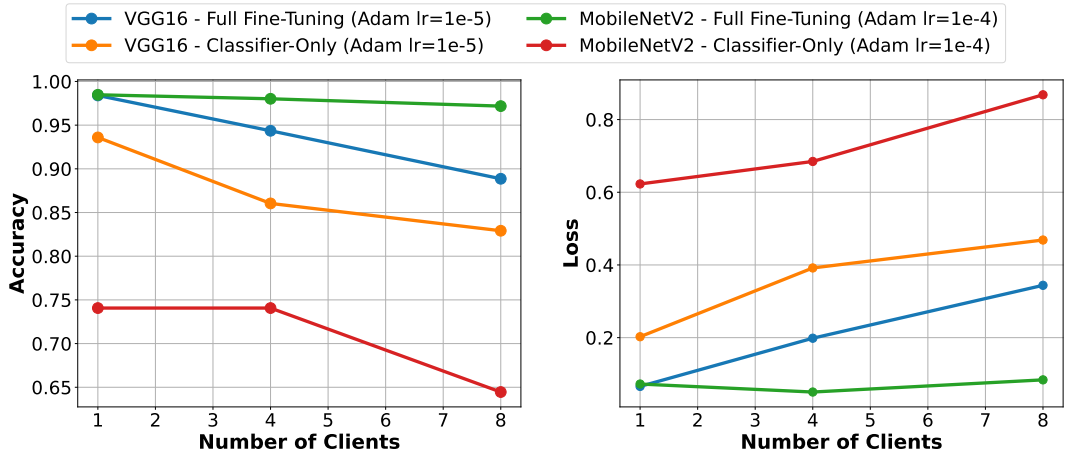


Figure 4.20: Impact of client number on federated learning with IID partitions

In general, increasing the number of clients leads to higher loss and lower accuracy. Nevertheless, full fine-tuning consistently achieves better performance, maintaining higher accuracy and lower loss than classifier-only fine-tuning, even in the federated setting.

For a rough comparison between centralized and federated performance, we consider the single-client case. Based on the average accuracies reported in Table 4.4, the results differ by less than 1%, except for MobileNet under classifier-only fine-tuning, which exhibits a 7% decrease. However, this comparison should be interpreted with caution, as the experimental settings are not directly comparable: in the federated scenario, 20% of the training set is reserved for validation, training is performed over 20 communication rounds instead of 10 epochs, and the learning rates differ.

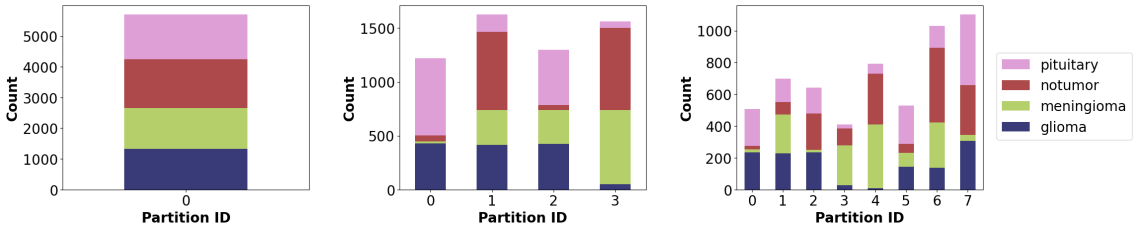
Model	Training Strategy	# Clients	Accuracy	Loss
VGG16	Full Fine-Tuning (Adam lr=1e−5)	1	0.984	0.066
		4	0.944	0.198
		8	0.889	0.344
	Classifier-Only (Adam lr=1e−5)	1	0.936	0.203
		4	0.860	0.392
		8	0.829	0.469
MobileNetV2	Full Fine-Tuning (Adam lr=1e−4)	1	0.985	0.072
		4	0.980	0.050
		8	0.972	0.084
	Classifier-Only (Adam lr=1e−4)	1	0.741	0.623
		4	0.741	0.685
		8	0.645	0.868

Table 4.10: Federated learning performance of VGG16 and MobileNetV2 under different training strategies and client numbers with IID data partitioning.

A final consideration concerns the choice of learning rate. Based on preliminary experiments (not reported in this thesis), **lower learning rates** were adopted in the federated setting compared to centralized training. This adjustment is necessary to prevent local updates from causing excessive divergence in the global model parameters. In particular, higher learning rates can destabilize local updates, especially under non-IID data distributions, the next scenario we analyze, thereby increasing the gap between federated and centralized performance and complicating convergence. Nevertheless, the learning rates used here remain within the range previously shown to support high accuracy in the centralized attack scenario (Figure 4.17).

4.4.2.3 Non-IID Partitioning

Non-IID partitioning refers to scenarios where data distributions differ significantly across clients. For instance, one client may have mostly samples of the `notumor` class while another may have very few or none at all. This heterogeneity reflects more realistic situations in medical practice, where certain conditions are more or less prevalent depending on the population of a specific area.



(a) 1 Client Partition **(b)** 4 Clients Partition **(c)** 8 Clients Partition

Figure 4.21: Non-IID partitioning using Dirichlet distribution sampling with $\alpha = 1.0$

As expected, also in the non-IID setting using the partitions shown in Figure 4.21, increasing the number of clients leads to lower accuracy and higher loss, as reported in Figure 4.22 and Table 4.11.

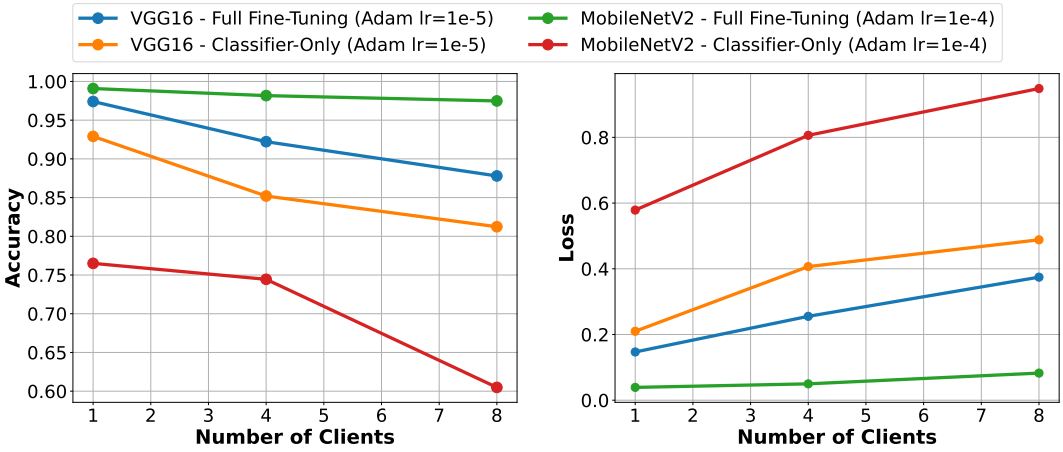


Figure 4.22: Impact of client number on federated learning with Non-IID partitions

Model	Training Strategy	# Clients	Accuracy	Loss
VGG16	Full Fine-Tuning (Adam lr=1e−5)	1	0.974	0.147
		4	0.922	0.255
		8	0.878	0.375
	Classifier-Only (Adam lr=1e−5)	1	0.929	0.210
		4	0.852	0.407
		8	0.812	0.488
MobileNetV2	Full Fine-Tuning (Adam lr=1e−4)	1	0.991	0.039
		4	0.982	0.050
		8	0.975	0.083
	Classifier-Only (Adam lr=1e−4)	1	0.765	0.579
		4	0.744	0.806
		8	0.605	0.949

Table 4.11: Federated learning performance of VGG16 and MobileNetV2 under different training strategies and client numbers with non-IID data partitioning.

Although this configuration more closely approximates real-world federated learning

scenarios, it also introduces additional challenges, particularly for aggregation strategies such as `FedAvg`, which have been shown to struggle under heterogeneous client distributions.

However, despite these limitations, the accuracies obtained in the non-IID experiments above remain very close to the IID case, making them acceptable in practice. This relative robustness may be explained by the use of pretrained models.

In typical federated learning scenarios, models are often initialized randomly and trained from scratch, a process that can significantly slow convergence when client data are non-IID. By contrast, initializing from pretrained weights, common practice in centralized deep learning but less frequently adopted in federated settings, provides a more favorable starting point. As demonstrated in [48], pretraining can reduce the performance gap between federated and centralized training, stabilize global aggregation, and mitigate sensitivity to suboptimal averaging, thereby offering particular benefits under heterogeneous client distributions. If instead custom models trained from scratch had been used in this setup, we would expect slower convergence and larger performance degradation compared to the IID case.

4.5 Attacking the Federated Models

Having introduced the federated learning setup, we now extend our analysis to evaluate the impact of backdoor attacks in this distributed scenario. In particular, we investigate whether full fine-tuning continues to absorb the trigger while maintaining high accuracy, and whether classifier-only fine-tuning resists trigger learning, as observed in the centralized setting.

4.5.1 Attack Setup

The experimental configuration for federated learning remains consistent with the baseline setup described in Table 4.9. In this section, we additionally introduce adversarial clients that poison part of their local data before contributing updates to the global model, replicating the attack methodology from the centralized case but adapted to a federated setting with 4 clients. The parameters governing this attack are summarized in Table 4.12.

Parameter	**Description**
Number of Clients	A total of 4 clients participate in the training process.
Number of Malicious Clients	Varied across experiments (1, 2, 3 and 4) to evaluate different numbers of adversarial clients.
Poisoning Rate	Fraction of each malicious client's local training data that is poisoned, evaluated at 0.3, 0.6, and 0.9
Trigger	Same trigger design and placement as in the centralized experiments: 4×4 white square positioned in the bottom-right quadrant of the image, offset by a margin of 50 pixels from the edges.
Target Label	Class assigned to all triggered inputs during the attack is `notumor`.

Table 4.12: Federated backdoor attack parameters

The following two paragraphs report results from an extensive backdoor simulation study. We evaluate the attack under two partitioning strategies: IID (optimal case) and non-IID (realistic case). In total, **96 experiments** were conducted, equally split between the two partitioning schemes (48 each). These experiments span all combinations of two model architectures (VGG-16 and MobileNetV2), two training strategies (full fine-tuning and classifier-only), four levels of adversarial participation (1, 2, 3, and 4 malicious clients), and three poisoning rates per malicious client (30%, 60%, 90%).

For each experiment, all malicious clients in a given run use the same poisoning rate. For example, the configuration *two malicious clients, 60% poisoning* (equivalently denoted as $m = 2$, $pr = 0.6$) indicates that each of the two adversarial clients corrupts 60% of its local training data with the trigger. Under IID partitioning this corresponds to the trigger being present in a majority of the global training samples, whereas under non-IID partitioning its global prevalence depends on the relative sizes of the poisoned clients' datasets.

As in the previous federated experiments, 20% of each client's local data is reserved for validation (serving as a local test set for the client). This results in a total of **4568 training samples** per federated simulation, slightly fewer than in the centralized case (5712 samples). However, this difference is not expected to meaningfully affect the subsequent observations.

4.5.2 Attack Evaluation under IID Partitions (Optimal Case)

In this optimal configuration, the dataset follows an IID partitioning scheme, as depicted in Figure 4.19b. Summary plots of the main findings are presented below, while the complete numerical results are provided in the Appendix in Tables A.1 and A.2.

Note that CAD values are computed relative to the baseline results reported in Table 4.10, where fully fine-tuned VGG-16 and MobileNetV2 with four clients achieved 0.944 and 0.980 accuracy, respectively, and classifier-only fine-tuned VGG-16 and MobileNetV2 achieved 0.860 and 0.741 accuracy, respectively, under the same conditions.

4.5.2.1 Full Fine-Tuning Strategy

We recall that full fine-tuning denotes the setting in which all model parameters are updated during training.

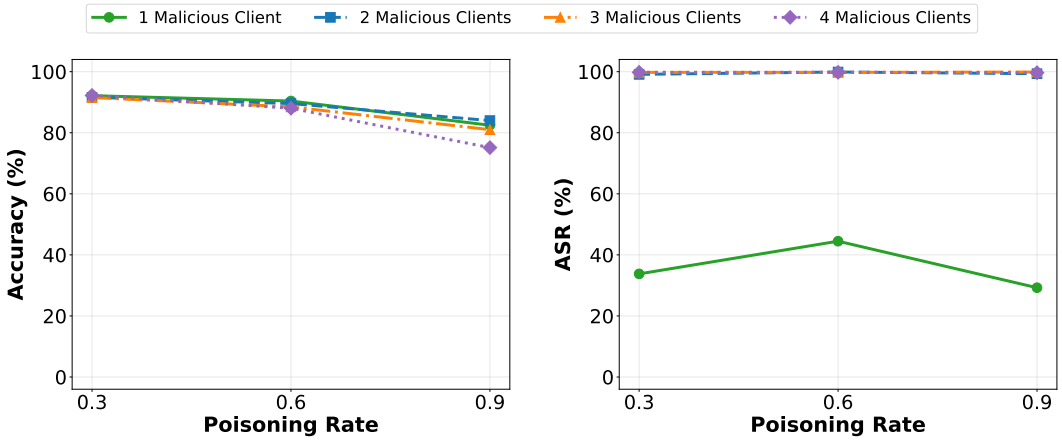


Figure 4.23: Accuracy and ASR of backdoor attacks on *IID Federated VGG-16* under full fine-tuning with varying numbers of adversaries and poisoning rates

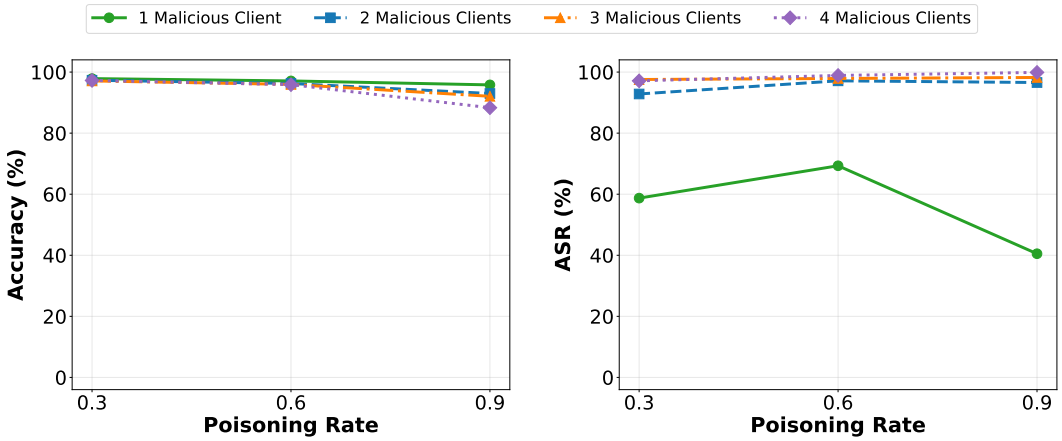


Figure 4.24: Accuracy and ASR of backdoor attacks on *IID Federated MobileNetV2* under full fine-tuning with varying numbers of adversaries and poisoning rates

Accuracy The accuracy of VGG-16 remains high across most scenarios, ranging from 0.810 to 0.921, except in the case of 4 malicious clients each poisoning 90% of their local data, where it drops to 0.751 on the clean test set. This is still remarkable given that 90% of the global training set is poisoned (4109 of 4568 samples). The corresponding clean accuracy drop (CAD) ranges from 0.022 ($m = 1$, $pr = 0.3$) up to 0.192 ($m = 4$, $pr = 0.9$).

Under both normal and attack conditions, Federated MobileNetV2 with full fine-tuning proves to be more accurate than Federated VGG-16, achieving accuracies decreasing from 0.979 with ($m = 1$, $pr = 0.3$) to 0.883 with ($m = 4$, $pr = 0.9$), and consistently maintaining this superiority across all combinations of adversarial clients and poisoning rates. The corresponding CAD values are substantially lower, spanning from 0.002 to 0.097.

Attack Success Rate (ASR) Excluding the scenarios with a single malicious client, VGG-16 consistently achieves approximately 99% ASR whenever there are two, three, or four malicious clients, independent of the poisoning rate. In contrast, excluding the single-adversary case, MobileNet reaches 99% ASR only when $m = 4$ and $pr = 0.9$; in the other cases, its ASR lies between 92.8% and 98.9%, still indicating very high attack effectiveness.

Comparison with the Centralized Setting The general behavior observed in the centralized environment is the same as that resulting from this federated experiment: fully fine-tuned models are able to learn both the trigger and medical features with ease.

In the centralized setting, poisoning just 10% of the data under full fine-tuning was sufficient to maintain high accuracy while achieving near-perfect ASR (Section 4.3.2). Moreover, as expected, higher poisoning rates of 30%, 60%, and 90% maintained high ASR while only slightly reducing accuracy, as shown in Figure 4.18.

In the federated setting, however, the situation is very different. Considering the case of a single malicious client, the adversary injects 342 ($pr = 0.3$), 684 ($pr = 0.6$), or 1026 ($pr = 0.9$) poisoned samples out of 4568 into the training process, corresponding to 7.5%, 15%, and 22.5% of the global dataset, respectively. Despite this seemingly high fraction, the ASR remains far from perfect in all three cases.

This difference highlights the inherent robustness of federated learning compared to centralized training. Since poisoned samples are restricted to a single client, their effect is naturally mitigated during aggregation through `FedAvg`, particularly when the majority of clients behave honestly. Even if the local model of a client is heavily poisoned, the corresponding gradient updates are down-weighted by the total number of clients, and their impact on the global model remains limited.

Interestingly, in the single-malicious-client scenario, the ASR rises from $pr = 0.3$ to $pr = 0.6$ but drops at $pr = 0.9$. This counterintuitive behavior may stem from `FedAvg` aggregation: at very high poisoning rates, the malicious client's update can strongly conflict with honest updates, which, under an IID data distribution, partially offsets the poisoned effect. Consequently, moderate poisoning rates can propagate the backdoor more effectively than excessively high rates.

4.5.2.2 Classifier-Only Fine-Tuning Strategy

We recall that classifier-only fine-tuning involves freezing all convolutional layers while updating only the final classification layer.

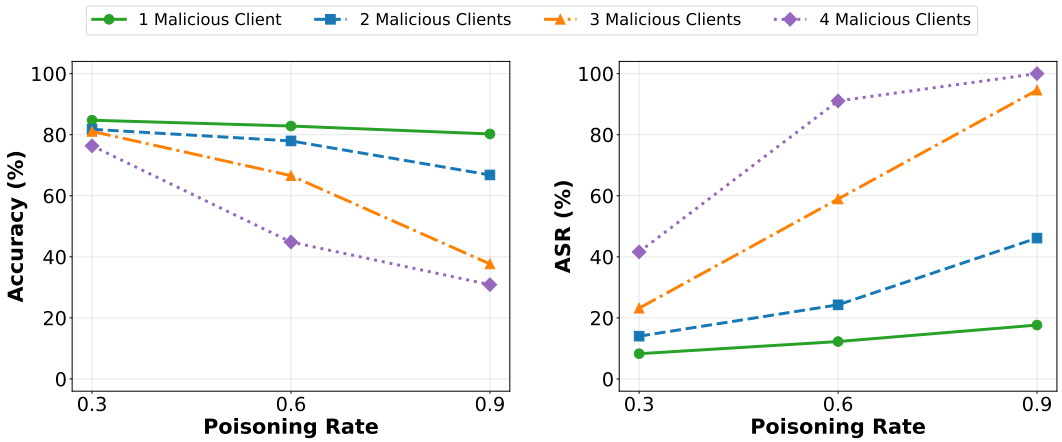


Figure 4.25: Accuracy and ASR of backdoor attacks on *IID Federated VGG-16* under classifier-only fine-tuning with varying numbers of adversaries and poisoning rates

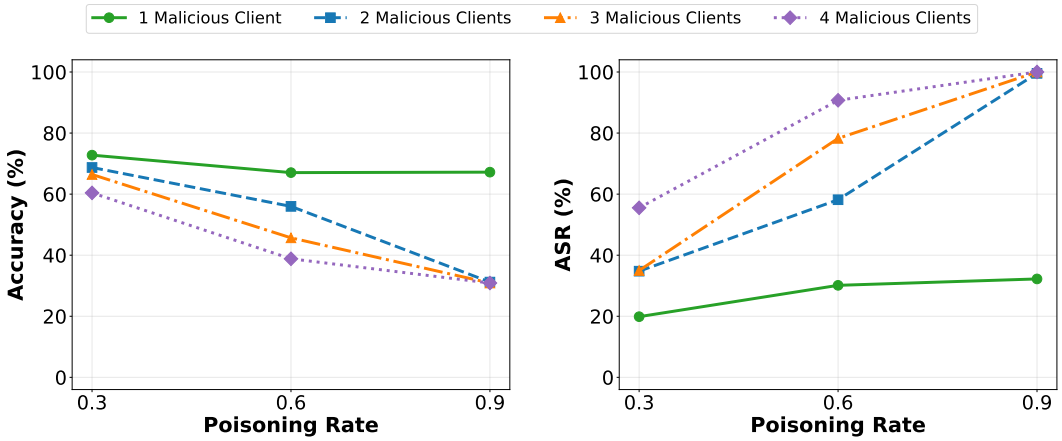


Figure 4.26: Accuracy and ASR of backdoor attacks on *IID Federated MobileNetV2* under classifier-only fine-tuning with varying numbers of adversaries and poisoning rates

Accuracy For VGG-16, accuracy drops from 0.847 (CAD 0.013) with $m = 1$ at $pr = 0.3$ to 0.309 (CAD 0.551) with $m = 4$ at $pr = 0.9$. Similarly, MobileNetV2 ranges from 0.728 (CAD 0.013) to 0.309 (CAD 0.432) across the same scenarios.

Both models maintain high resilience with a single malicious client, exhibiting very small CAD values. However, performance deteriorates as soon as two malicious clients are present, although the models continue to resist learning the trigger better than in the full fine-tuning case.

ASR An interesting observation, not apparent in the previous experiment, is that the ASR exceeds 50% in VGG-16 with $m = 3$ at $pr = 0.6$, and in MobileNetV2 with $m = 2$ at $pr = 0.6$. However, these higher ASR values are accompanied by reduced accuracy and increased CADs: 0.666 accuracy and CAD of 0.195 for VGG-16, and 0.560 accuracy and CAD of 0.181 for MobileNetV2.

Another noteworthy aspect is that this threshold is surpassed in four configurations for VGG-16, whereas it occurs seven times for MobileNetV2, indicating a greater sensitivity of MobileNetV2 to various combinations of poisoning rates and malicious client counts.

Comparison with the Centralized Setting The general behavior observed in the centralized environment is the same as that resulting from this federated experiment: classifier-only fine-tuned models exhibit a strong inverse correlation between accuracy and ASR, such that higher accuracy corresponds to lower ASR, and conversely, lower accuracy corresponds to higher ASR. For instance, scenarios with 0.309 accuracy, representing the minimum achievable accuracy since all `notumor` samples are correctly classified, correspond to an ASR of 100%.

To compare these results with the centralized setting, Figure 4.18 shows that classifier-only models in centralized training reach $ASR > 50\%$ at approximately a 0.4 poisoning rate. In the federated setting, however, VGG-16 reaches this threshold when 2,055 out of 4,568 samples are poisoned ($\approx 45\%$ of the global dataset), whereas MobileNetV2 reaches it at 1,370/4,568 samples ($\approx 30\%$), confirming that a very similar behavior is observed.

4.5.3 Attack Evaluation under Non-IID Partitions (Realistic Case)

Experiments conducted with an IID distribution, where all clients possess approximately the same number of samples, are often overly simplistic. In such cases, detecting malicious clients can be relatively straightforward, as honest participants tend to send similar gradients while malicious ones deviate significantly. Naturally, detection becomes more difficult when the majority of clients are malicious.

In Non-IID settings (Figure 4.21b), the uneven distribution of tumor classes across clients means that even updates from two honest clients can differ substantially due to data heterogeneity, making it challenging to distinguish between benign and malicious updates. In the following, we extend our evaluation to this scenario to determine whether our previous observations remain valid and whether comparisons with the IID case are still meaningful, providing insight into model behavior under a more realistic federated setting.

Summary plots of the main findings are provided below, with complete numerical results in Tables A.3 and A.4. CAD values are computed relative to the baselines in Table 4.11,

where fully fine-tuned VGG-16 and MobileNetV2 with four clients achieved 0.922 and 0.982 accuracy, respectively, and classifier-only fine-tuned VGG-16 and MobileNetV2 achieved 0.852 and 0.744.

4.5.3.1 Full Fine-Tuning Strategy

Observing the curves in the plots, the general behavior of the Non-IID federated models trained with full fine-tuning closely resembles that observed in the IID case. ASR is very high, except in the scenario with a single adversarial client, reaching near-perfect values for VGG-16, while MobileNetV2 shows more distinct curves depending on the number of adversaries. At the same time, accuracies decrease in proportion to the number of poisoned samples and malicious participants in both models.

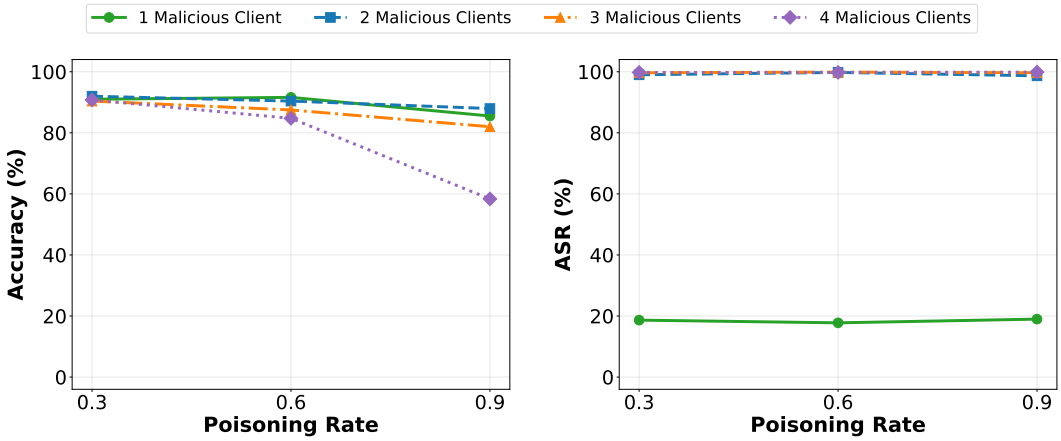


Figure 4.27: Accuracy and ASR of backdoor attacks on *Non-IID Federated VGG-16* under full fine-tuning with varying numbers of adversaries and poisoning rates

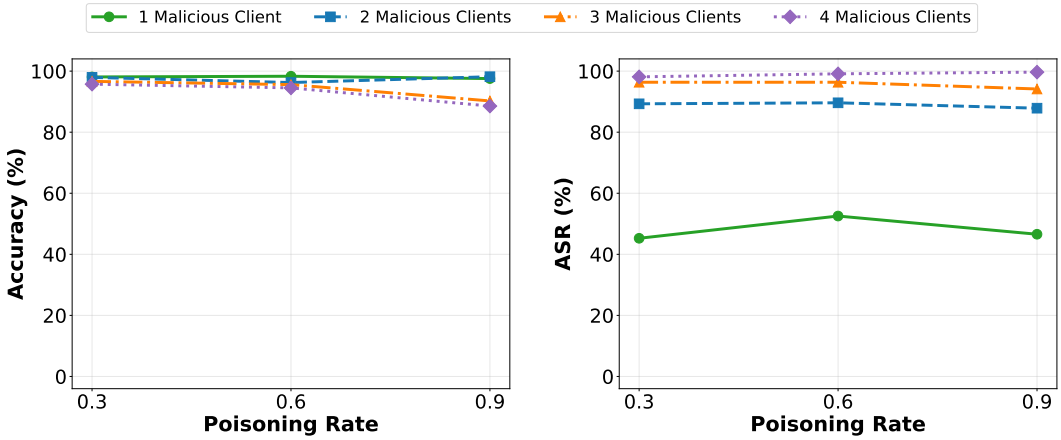


Figure 4.28: Accuracy and ASR of backdoor attacks on *Non-IID Federated MobileNetV2* under full fine-tuning with varying numbers of adversaries and poisoning rates

Interestingly, except for the scenarios with four adversarial clients, the CAD in the Non-IID setting is lower than in the IID case. This may be explained by the slightly lower baseline accuracy of the Non-IID models; nevertheless, it implies that under realistic, non-homogeneous partitions, the accuracy drop after backdoor installation remains limited when models are fully fine-tuned.

These minor differences do not alter the expected behavior: experiments confirm that under Non-IID partitions, when full fine-tuning is used and at least two clients are malicious, the trigger is learned while medical features are preserved.

4.5.3.2 Classifier-Only Fine-Tuning Strategy

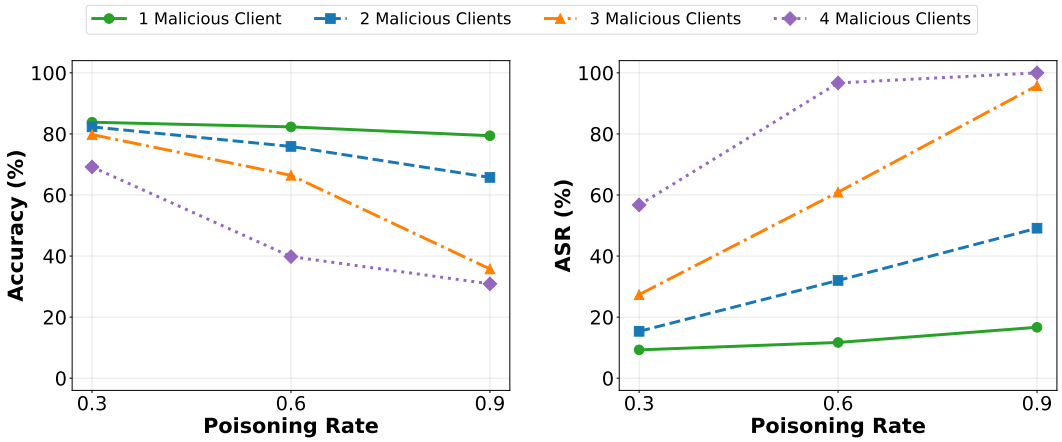


Figure 4.29: Accuracy and ASR of backdoor attacks on *Non-IID Federated VGG-16* under classifier-only fine-tuning with varying numbers of adversaries and poisoning rates

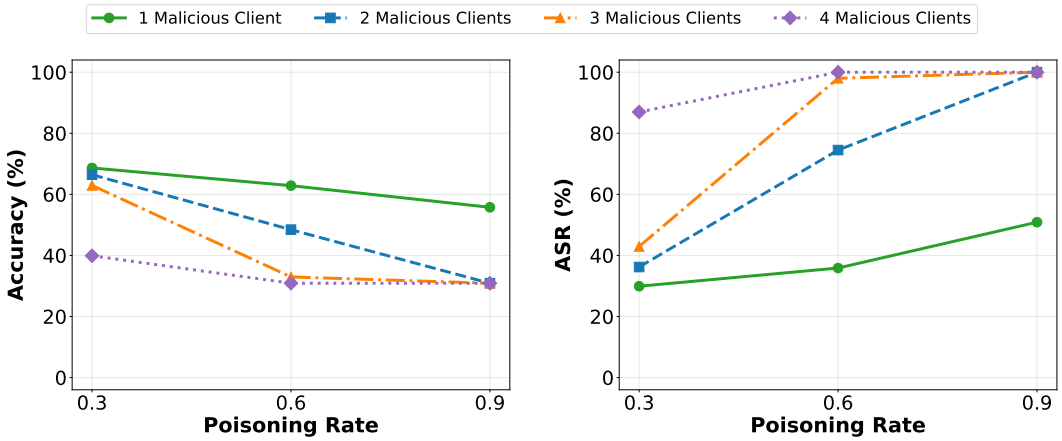


Figure 4.30: Accuracy and ASR of backdoor attacks on *Non-IID Federated MobileNetV2* under classifier-only fine-tuning with varying numbers of adversaries and poisoning rates

In the classifier-only setting, CAD and ASR values for VGG-16 are similar between IID and Non-IID partitions, whereas MobileNet appears slightly more vulnerable under Non-IID conditions, exhibiting larger CADs. Notably, MobileNet reaches 100% ASR in only two IID cases, compared to four Non-IID cases.

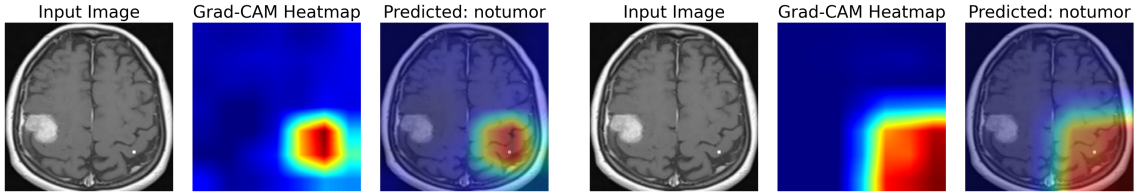
Beyond these differences, the familiar behavior remains evident under Non-IID conditions: as ASR increases, accuracy decreases, and vice versa.

4.5.4 Defending Federated Learning with Explainable AI

It has been shown that, in a centralized model, explainable AI techniques such as Grad-CAM can be used at inference time to detect the presence of a trigger in an image. This approach is also applicable in the federated scenario. After the collaborative training phase,

the global model is deployed to each client. If a client injects a backdoor during training, any input containing the trigger can be misclassified across all clients in the federated network during inference. Grad-CAM can then reveal that the model's attention is focused on the trigger region rather than the medically relevant areas, highlighting the presence of a backdoor in the model.

To illustrate this, we used the Non-IID partitioning shown in Figure 4.21b and conducted two additional experiments with four clients under full fine-tuning, setting $m = 2$ and $pr = 0.5$ for both models. Results in Figure 4.31 show that both models misclassify a meningioma MRI as `notumor`. The corresponding Grad-CAM visualizations make the presence of the backdoor trigger immediately evident to a human observer and could also be leveraged to generate a mask, which may then be used by a GAN or another CNN to automatically detect and potentially correct the trigger.



(a) Federated VGG-16
Accuracy: 0.903 - ASR: 99.6%

(b) Federated MobileNetV2
Accuracy: 0.967 - ASR: 91.6%

Figure 4.31: Grad-CAM for a meningioma MRI with trigger in a federated setting

Note that in all our experiments, the trigger is visible, allowing a human observer to directly notice and suspect the 4×4 grid without relying on the Grad-CAM heatmap. However, this visible trigger was chosen solely to simplify the poisoning process. Previous studies have demonstrated that adversaries can craft triggers imperceptible to humans, indicating that visual inspection alone is insufficient as a defense.

4.5.5 Final Considerations

Key Takeaways

Observations from the centralized environment generally extend to federated settings under both IID and Non-IID conditions, with only minor performance variations.

When using a trusted pretrained model (i.e., one that does not contain any backdoors) and fine-tuning it on a task with a partially untrusted training set, adopting a classifier-only fine-tuning strategy can help *mitigate* the influence of potential triggers in the data. This approach is effective in both centralized and distributed environments. Furthermore, this strategy offers a simple yet informative *detection mechanism*: when a backdoor trigger is learned, model accuracy typically drops noticeably. This phenomenon reflects the inverse correlation between ASR and accuracy, previously observed in centralized settings and shown here to also hold in the federated context.

However, pretrained models and classifier-only fine-tuning are not always suitable for highly specialized tasks, as overall performance may otherwise degrade. In such cases, full fine-tuning should be employed. Under this strategy, both task-relevant features and potential backdoor triggers are learned easily when at least two malicious clients are present,

thereby necessitating alternative defenses. When only a single client is malicious, the trigger can still be learned, but the ASR typically remains low.

As in the centralized setting, explainability methods such as Grad-CAM can be employed by each client at inference time to assess whether a trigger is present in the input image and whether the shared model has been compromised. When human oversight is available, these visual explanations can assist in identifying suspicious activations. Alternatively, automated systems could leverage Grad-CAM generated masks to detect or even remove potential triggers without manual intervention.

Chapter 5

Conclusion

In this thesis, after selecting two pre-trained models based on their architecture and performance in the classification task, an extensive series of experiments was conducted to gain a deeper understanding of how backdoor attacks operate.

The primary objective was to determine whether variations in models, training strategies (full or classifier-only fine-tuning), training environments (centralized or distributed), attack and training parameters, and data partitioning schemes could make backdoor attacks more detectable or less effective in these settings. Ultimately, the aim was to identify principles that could guide the design of more generalizable and robust backdoor defense mechanisms applicable to both federated and centralized learning frameworks.

5.1 Main Findings

A subtle modification of just a few pixels in a portion of the training samples can successfully implant a backdoor into a medical screening system, leading the model to misclassify a brain tumor image as a non-tumor case.

In the centralized setting, the training strategy was found to have a significant impact on the effectiveness of backdoor attacks in pre-trained models. Despite architectural differences between VGG-16 and MobileNetV2, particularly in their convolutional and feature extraction layers, the trigger learning process remained consistent when the same training strategy was applied.

Specifically, when pre-trained models were *fully fine-tuned*, using their original weights merely as initialization, the models consistently learned both the medical features and the trigger pattern. In one of the initial experiments, poisoning only 10% of the training samples was sufficient to achieve an ASR of 98.90% for MobileNetV2 and 99.89% for VGG-16 on the poisoned test set, while maintaining high clean accuracy. This result indicates that implanting a trigger into the convolutional layers through dataset poisoning is relatively easy when this fine-tuning strategy is applied.

Another training strategy involves starting from a trusted pre-trained model that performs well on the target task and freezing its feature extractor while *fine-tuning only the classifier*. This approach has been shown to *mitigate* the impact of potential backdoor attacks embedded within the training data.

With this training strategy, a low poisoning rate results in a lower ASR compared to

full fine-tuning, as the trigger is not effectively learned. For example, poisoning 10% of the training samples produced ASRs of 30.68% and 9.27%, with corresponding clean accuracies of 78.57% and 89.24% for MobileNetV2 and VGG-16, respectively.

When the poisoning rate increases, the model begins to learn the trigger more effectively, leading to higher ASRs but a noticeable decline in clean accuracy, which makes *anomalies easier to detect*. For instance, achieving an ASR above 80% required poisoning 60% of the training data, at which point the clean accuracy dropped to approximately 50% for both models.

In the federated setting, we demonstrated that pre-trained models can be effectively employed and achieve performance comparable to the centralized scenario. Under normal (non-attack) conditions with IID client partitions and `FedAVG` aggregation, accuracies differed from the centralized setting by less than 1%, except for MobileNet under classifier-only fine-tuning, which showed a 7% decrease.

In the attack scenario, we observed that the general behavior in the federated setting closely resembles that in the centralized environment. Federated VGG-16 and Federated MobileNetV2 exhibit very similar patterns, with the primary differences in results once again arising from the chosen fine-tuning strategy.

In particular, when exposed to backdoor attacks, and *full fine-tuning* is applied with at least two malicious clients, the trigger is consistently learned while medical features are preserved, similar to the behavior observed in the centralized setting. In the IID tests, VGG-16 consistently achieved approximately 99% ASR regardless of whether there were two, three, or four malicious clients, independent of the poisoning rate, while MobileNetV2 ranged between 92.8% and 99%.

The case of a single malicious client provides another insight: when considering the total number of poisoned samples, federated learning is generally more robust than centralized learning as long as most clients are honest. Even when the adversary poisoned 7.5%, 15%, or 22.5% of the global dataset, the ASR remained far from perfect, whereas in the centralized setting, poisoning just 10% was sufficient to achieve a near-perfect ASR under the same training strategy.

In contrast, *classifier-only fine-tuned* models, as in the centralized setting, exhibit strong resilience to trigger absorption. If the model is forced to learn the trigger by poisoning a larger proportion of samples, the accuracy on the clean test set decreases. Thus, in both centralized and federated settings, an inverse correlation exists between ASR and clean accuracy when this training strategy is employed. This suggests that measuring accuracy on a test set known to be free of poisoned images can serve as a *detection* technique: the presence of a backdoor manifests as reduced accuracy.

In addition to the ideal IID case, we evaluated non-IID partitions to simulate more realistic scenarios and found that data partitioning had no substantial impact on pre-trained models or the observations above, under either normal or attack conditions.

To design a robust backdoor defense applicable to both centralized and distributed environments, we cannot rely solely on classifier-only fine-tuning of pre-trained models, even though this strategy can be applied in both scenarios, because some classification tasks may not achieve high accuracy with pre-trained models.

For this reason, this work also highlights the potential of explainability techniques, specifically Grad-CAM, not only to help clinicians better understand model predictions and increase transparency, but also to *detect potential attacks* by revealing when the CNN focuses on irrelevant image regions (i.e., triggers) rather than medically significant areas.

These two techniques can be employed separately or in combination, providing a solid foundation to mitigate and detect the presence of backdoor attacks, and serving as a starting point for the development of more comprehensive defense mechanisms.

5.2 Future Works

Future directions for this work are numerous and span multiple directions.

From a *dataset and model perspective*, one direction is to enlarge the dataset by creating pipelines that, for example, extract images directly from DICOM files. Another approach is to enhance the data augmentation process and employ GAN-based synthetic image generation, thereby increasing the number of training samples. Additionally, future work could explore different application contexts by testing the methodology on other classification tasks within the medical domain or extending it to non-medical image datasets, to determine whether the observed behaviors are specific to medical imaging or more general. Although this study analyzed two very different architectures, further evaluation of additional pre-trained models could help assess whether these behaviors generalize across architectures.

From an *attack* perspective, numerous experiments could be conducted to evaluate how attack effectiveness varies with changes in trigger size, visibility, or position, as well as with different poisoning strategies. Untargeted attacks or targeted attacks with alternative labels could also be explored. Other forms of backdoor attacks, such as invisible triggers or noise-based triggers, can be investigated to assess whether the proposed detection techniques (classifier-only fine-tuning and Grad-CAM) remain effective.

Future research could also explore more advanced and robust *defense mechanisms*, including the combination of multiple techniques, such as classifier-only fine-tuning, Grad-CAM, and model pruning (not covered in this thesis but a promising direction), to enhance resilience against backdoor attacks. Additionally, developing an autonomous pipeline that automatically analyzes Grad-CAM results to detect triggers without human intervention could enable real-time detection and further strengthen defenses.

In the specific context of federated learning, *privacy threats* remain a critical concern. Integrating mechanisms such as secure aggregation or differential privacy, and testing their interaction with the proposed methods, could complement backdoor defense strategies while preserving data confidentiality.

More broadly, progress in this field will depend on close collaboration between engineers, researchers, and data scientists. Building unified defense mechanisms that work effectively in both centralized and federated settings would be an important step toward AI systems that are not only safer, but also truly reliable in real-world applications.

Appendix A

Extended Results

Table A.1: Attack results in the IID setting for Federated VGG-16 under different numbers of malicious clients and poisoning rates

Model	Training Strategy	# Mal. Clients	Poisoning Rate	Poisoned Samples per Client	Clean Accuracy	CAD	ASR (%)
VGG-16	Full Fine-Tuning (Adam lr=1e−5)	1	0.3	342/1140, ~~0/1142~~, ~~0/1142~~, ~~0/1144~~	0.921	0.022	33.77
			0.6	684/1140, ~~0/1142~~, ~~0/1142~~, ~~0/1144~~	0.904	0.040	44.48
			0.9	1026/1140, ~~0/1142~~, ~~0/1142~~, ~~0/1144~~	0.825	0.119	29.25
		2	0.3	342/1140, ~~0/1142~~, ~~0/1142~~, 343/1144	0.916	0.027	99.12
			0.6	684/1140, ~~0/1142~~, ~~0/1142~~, 686/1144	0.896	0.048	99.89
			0.9	1026/1140, ~~0/1142~~, ~~0/1142~~, 1029/1144	0.840	0.104	99.34
		3	0.3	342/1140, 342/1142, ~~0/1142~~, 343/1144	0.915	0.028	99.78
			0.6	684/1140, 685/1142, ~~0/1142~~, 686/1144	0.885	0.059	99.78
			0.9	1026/1140, 1027/1142, ~~0/1142~~, 1029/1144	0.810	0.133	99.89
		4	0.3	342/1140, 342/1142, 342/1142, 343/1144	0.922	0.021	99.78
			0.6	684/1140, 685/1142, 685/1142, 686/1144	0.880	0.063	99.78
			0.9	1026/1140, 1027/1142, 1027/1142, 1029/1144	0.751	0.192	99.67
	Classifier-Only (Adam lr=1e−5)	1	0.3	342/1140, ~~0/1142~~, ~~0/1142~~, ~~0/1144~~	0.847	0.013	8.28
			0.6	684/1140, ~~0/1142~~, ~~0/1142~~, ~~0/1144~~	0.828	0.032	12.25
			0.9	1026/1140, ~~0/1142~~, ~~0/1142~~, ~~0/1144~~	0.802	0.058	17.66
		2	0.3	342/1140, ~~0/1142~~, ~~0/1142~~, 343/1144	0.818	0.043	14.02
			0.6	684/1140, ~~0/1142~~, ~~0/1142~~, 686/1144	0.780	0.081	24.28
			0.9	1026/1140, ~~0/1142~~, ~~0/1142~~, 1029/1144	0.668	0.192	46.14
		3	0.3	342/1140, 342/1142, ~~0/1142~~, 343/1144	0.811	0.050	23.18
			0.6	684/1140, 685/1142, ~~0/1142~~, 686/1144	0.666	0.195	58.94
			0.9	1026/1140, 1027/1142, ~~0/1142~~, 1029/1144	0.377	0.484	94.59
		4	0.3	342/1140, 342/1142, 342/1142, 343/1144	0.764	0.097	41.61
			0.6	684/1140, 685/1142, 685/1142, 686/1144	0.449	0.412	91.06
			0.9	1026/1140, 1027/1142, 1027/1142, 1029/1144	0.309	0.551	100.0

Table A.2: Attack results in the IID setting for Federated MobileNetV2 under different numbers of malicious clients and poisoning rates

Model	Training Strategy	# Mal. Clients	Poisoning Rate	Poisoned Samples per Client	Clean Accuracy	CAD	ASR (%)
MobileNetV2	Full Fine-Tuning (Adam lr=1e−4)	1	0.3	342/1140, ~~0/1142~~, ~~0/1142~~, ~~0/1144~~	0.979	0.002	58.72
			0.6	684/1140, ~~0/1142~~, ~~0/1142~~, ~~0/1144~~	0.971	0.009	69.32
			0.9	1026/1140, ~~0/1142~~, ~~0/1142~~, ~~0/1144~~	0.963	0.022	40.51
		2	0.3	342/1140, ~~0/1142~~, ~~0/1142~~, 343/1144	0.973	0.007	92.83
			0.6	684/1140, ~~0/1142~~, ~~0/1142~~, 686/1144	0.963	0.018	97.13
			0.9	1026/1140, ~~0/1142~~, ~~0/1142~~, 1029/1144	0.930	0.050	96.58
		3	0.3	342/1140, 342/1142, ~~0/1142~~, 343/1144	0.971	0.009	97.57
			0.6	684/1140, 685/1142, ~~0/1142~~, 686/1144	0.960	0.021	97.90
			0.9	1026/1140, 1027/1142, ~~0/1142~~, 1029/1144	0.921	0.059	98.23
		4	0.3	342/1140, 342/1142, 342/1142, 343/1144	0.973	0.008	97.13
			0.6	684/1140, 685/1142, 685/1142, 686/1144	0.959	0.021	98.90
			0.9	1026/1140, 1027/1142, 1027/1142, 1029/1144	0.883	0.097	99.89
	Classifier-Only (Adam lr=1e−4)	1	0.3	342/1140, ~~0/1142~~, ~~0/1142~~, ~~0/1144~~	0.728	0.013	19.87
			0.6	684/1140, ~~0/1142~~, ~~0/1142~~, ~~0/1144~~	0.670	0.070	30.13
			0.9	1026/1140, ~~0/1142~~, ~~0/1142~~, ~~0/1144~~	0.672	0.069	32.23
		2	0.3	342/1140, ~~0/1142~~, ~~0/1142~~, 343/1144	0.687	0.053	34.77
			0.6	684/1140, ~~0/1142~~, ~~0/1142~~, 686/1144	0.560	0.181	58.17
			0.9	1026/1140, ~~0/1142~~, ~~0/1142~~, 1029/1144	0.312	0.429	99.56
		3	0.3	342/1140, 342/1142, ~~0/1142~~, 343/1144	0.664	0.076	34.99
			0.6	684/1140, 685/1142, ~~0/1142~~, 686/1144	0.457	0.284	78.26
			0.9	1026/1140, 1027/1142, ~~0/1142~~, 1029/1144	0.309	0.432	100.0
		4	0.3	342/1140, 342/1142, 342/1142, 343/1144	0.604	0.137	55.52
			0.6	684/1140, 685/1142, 685/1142, 686/1144	0.388	0.352	90.73
			0.9	1026/1140, 1027/1142, 1027/1142, 1029/1144	0.309	0.432	100.0

Table A.3: Attack results in the Non-IID setting for Federated VGG-16 under different numbers of malicious clients and poisoning rates

Model	Training Strategy	# Mal. Clients	Poisoning Rate	Poisoned Samples per Client	Clean Accuracy	CAD	ASR (%)
VGG-16	Full Fine-Tuning (Adam lr=1e−5)	1	0.3	292/976, ~~0/1302~~, ~~0/1039~~, ~~0/1251~~	0.910	0.012	18.65
			0.6	585/976, ~~0/1302~~, ~~0/1039~~, ~~0/1251~~	0.916	0.006	17.77
			0.9	878/976, ~~0/1302~~, ~~0/1039~~, ~~0/1251~~	0.855	0.067	18.98
		2	0.3	292/976, ~~0/1302~~, ~~0/1039~~, 375/1251	0.920	0.002	99.01
			0.6	585/976, ~~0/1302~~, ~~0/1039~~, 750/1251	0.904	0.018	99.78
			0.9	878/976, ~~0/1302~~, ~~0/1039~~, 1125/1251	0.879	0.043	98.68
		3	0.3	292/976, 390/1302, ~~0/1039~~, 375/1251	0.904	0.018	99.67
			0.6	585/976, 781/1302, ~~0/1039~~, 750/1251	0.875	0.047	99.89
			0.9	878/976, 1171/1302, ~~0/1039~~, 1125/1251	0.820	0.102	99.67
		4	0.3	292/976, 390/1302, 311/1039, 375/1251	0.908	0.014	99.78
			0.6	585/976, 781/1302, 623/1039, 750/1251	0.847	0.075	99.78
			0.9	878/976, 1171/1302, 935/1039, 1125/1251	0.584	0.339	99.89
	Classifier-Only (Adam lr=1e−5)	1	0.3	292/976, ~~0/1302~~, ~~0/1039~~, ~~0/1251~~	0.838	0.014	9.27
			0.6	585/976, ~~0/1302~~, ~~0/1039~~, ~~0/1251~~	0.823	0.029	11.70
			0.9	878/976, ~~0/1302~~, ~~0/1039~~, ~~0/1251~~	0.794	0.058	16.67
		2	0.3	292/976, ~~0/1302~~, ~~0/1039~~, 375/1251	0.823	0.029	15.34
			0.6	585/976, ~~0/1302~~, ~~0/1039~~, 750/1251	0.759	0.093	32.01
			0.9	878/976, ~~0/1302~~, ~~0/1039~~, 1125/1251	0.658	0.195	49.12
		3	0.3	292/976, 390/1302, ~~0/1039~~, 375/1251	0.798	0.054	27.37
			0.6	585/976, 781/1302, ~~0/1039~~, 750/1251	0.664	0.188	60.93
			0.9	878/976, 1171/1302, ~~0/1039~~, 1125/1251	0.358	0.494	95.81
		4	0.3	292/976, 390/1302, 311/1039, 375/1251	0.692	0.160	56.73
			0.6	585/976, 781/1302, 623/1039, 750/1251	0.398	0.454	96.69
			0.9	878/976, 1171/1302, 935/1039, 1125/1251	0.309	0.543	100.0

Table A.4: Attack results in the Non-IID setting for Federated MobileNetV2 under different numbers of malicious clients and poisoning rates

Model	Training Strategy	# Mal. Clients	Poisoning Rate	Poisoned Samples per Client	Clean Accuracy	CAD	ASR (%)
MobileNetV2	Full Fine-Tuning (Adam lr=1e−4)	1	0.3	292/976, ~~0/1302~~, ~~0/1039~~, ~~0/1251~~	0.981	0.001	45.25
			0.6	585/976, ~~0/1302~~, ~~0/1039~~, ~~0/1251~~	0.983	-0.002	52.54
			0.9	878/976, ~~0/1302~~, ~~0/1039~~, ~~0/1251~~	0.976	0.006	46.58
		2	0.3	292/976, ~~0/1302~~, ~~0/1039~~, 375/1251	0.979	0.002	89.29
			0.6	585/976, ~~0/1302~~, ~~0/1039~~, 750/1251	0.963	0.019	89.62
			0.9	878/976, ~~0/1302~~, ~~0/1039~~, 1125/1251	0.982	0.000	87.86
		3	0.3	292/976, 390/1302, ~~0/1039~~, 375/1251	0.966	0.015	96.36
			0.6	585/976, 781/1302, ~~0/1039~~, 750/1251	0.956	0.026	96.36
			0.9	878/976, 1171/1302, ~~0/1039~~, 1125/1251	0.902	0.079	94.15
		4	0.3	292/976, 390/1302, 311/1039, 375/1251	0.957	0.024	98.12
			0.6	585/976, 781/1302, 623/1039, 750/1251	0.945	0.037	99.12
			0.9	878/976, 1171/1302, 935/1039, 1125/1251	0.886	0.096	99.67
	Classifier-Only (Adam lr=1e−4)	1	0.3	292/976, ~~0/1302~~, ~~0/1039~~, ~~0/1251~~	0.686	0.058	29.91
			0.6	585/976, ~~0/1302~~, ~~0/1039~~, ~~0/1251~~	0.629	0.116	35.87
			0.9	878/976, ~~0/1302~~, ~~0/1039~~, ~~0/1251~~	0.558	0.187	50.88
		2	0.3	292/976, ~~0/1302~~, ~~0/1039~~, 375/1251	0.664	0.080	36.20
			0.6	585/976, ~~0/1302~~, ~~0/1039~~, 750/1251	0.484	0.260	74.50
			0.9	878/976, ~~0/1302~~, ~~0/1039~~, 1125/1251	0.309	0.436	100.0
		3	0.3	292/976, 390/1302, ~~0/1039~~, 375/1251	0.629	0.115	42.94
			0.6	585/976, 781/1302, ~~0/1039~~, 750/1251	0.330	0.415	98.01
			0.9	878/976, 1171/1302, ~~0/1039~~, 1125/1251	0.309	0.436	100.0
		4	0.3	292/976, 390/1302, 311/1039, 375/1251	0.399	0.346	86.98
			0.6	585/976, 781/1302, 623/1039, 750/1251	0.309	0.436	100.0
			0.9	878/976, 1171/1302, 935/1039, 1125/1251	0.309	0.436	100.0

Bibliography

[1] Charles Jt Butcher and Wajid Hussain. "Digital Healthcare: The Future". In: *Future Healthcare Journal* 9.2 (July 2022), pp. 113–117. ISSN: 25146645. DOI: 10.7861/fhj.2022-0046. URL: https://linkinghub.elsevier.com/retrieve/pii/S2514664524004855 (cit. on p. 1).

[2] European Commission. *Cybersecurity in healthcare – A healthcare sector resilient to cyber threats*. https://commission.europa.eu/cybersecurity-healthcare_en. Accessed: 2025-08-23. 2025 (cit. on p. 1).

[3] Pronnoy Dutta, Pradumn Upadhyay, Madhurima De, and R.G. Khalkar. "Medical Image Analysis using Deep Convolutional Neural Networks: CNN Architectures and Transfer Learning". In: *2020 International Conference on Inventive Computation Technologies (ICICT)*. 2020, pp. 175–180. DOI: 10.1109/ICICT48043.2020.9112469 (cit. on pp. 5, 20).

[4] Maryellen Giger. "Machine Learning in Medical Imaging". In: *Journal of the American College of Radiology* 15 (Feb. 2018). DOI: 10.1016/j.jacr.2017.12.028 (cit. on p. 6).

[5] Pranav Rajpurkar and Matthew P. Lungren. "The Current and Future State of AI Interpretation of Medical Images". In: *New England Journal of Medicine* 388.21 (May 25, 2023). Ed. by Jeffrey M. Drazen, Isaac S. Kohane, and Tze-Yun Leong, pp. 1981–1990. ISSN: 0028-4793, 1533-4406. DOI: 10.1056/NEJMra2301725. URL: http://www.nejm.org/doi/10.1056/NEJMra2301725 (cit. on p. 6).

[6] Abdulkader Helwan, Mohammad Khaleel Sallam Ma'aitah, Hani Hamdan, Dilber Uzun Ozsahin, and Ozum Tuncyurek. "Radiologists versus Deep Convolutional Neural Networks: A Comparative Study for Diagnosing COVID-19". In: *Computational and Mathematical Methods in Medicine* 2021 (May 10, 2021), p. 5527271. ISSN: 1748-670X. DOI: 10.1155/2021/5527271. PMID: 34055034. URL: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC8112196/ (cit. on p. 6).

[7] Xiaoxuan Liu et al. "A Comparison of Deep Learning Performance against Health-Care Professionals in Detecting Diseases from Medical Imaging: A Systematic Review and Meta-Analysis". In: *The Lancet Digital Health* 1.6 (Oct. 1, 2019), e271–e297. ISSN: 2589-7500. DOI: 10.1016/S2589-7500(19)30123-2. URL: https://www.sciencedirect.com/science/article/pii/S2589750019301232 (cit. on p. 6).

[8] Daniel McDuff et al. "Towards accurate differential diagnosis with large language models". In: *Nature* (2025), pp. 1–7 (cit. on p. 6).

[9] Harsha Nori et al. *Sequential Diagnosis with Language Models*. 2025. arXiv: 2506.22405 [cs.CL]. URL: https://arxiv.org/abs/2506.22405 (cit. on p. 6).

[10] Meghavi Rana and Megha Bhushan. "Machine Learning and Deep Learning Approach for Medical Image Analysis: Diagnosis to Detection". In: *Multimedia Tools and Applications* 82.17 (July 2023), pp. 26731–26769. ISSN: 1380-7501, 1573-7721. DOI: 10.1007/s11042-022-14305-w. URL: https://link.springer.com/10.1007/s11042-022-14305-w (cit. on p. 8).

[11] Alexander Amini, Ava Soleimany, et al. *MIT Introduction to Deep Learning*. https://introtodeeplearning.com/. Accessed: 2025-05-09. 2024. URL: https://introtodeeplearning.com/ (cit. on pp. 10, 15, 16, 18, 20).

[12] Rene Y Choi, Aaron S Coyner, Jayashree Kalpathy-Cramer, Michael F Chiang, and J Peter Campbell. "Introduction to Machine Learning, Neural Networks, and Deep Learning". In: () (cit. on p. 10).

[13] Zakaria Rguibi, Abdelmajid Hajami, Dya Zitouni, Amine Elqaraoui, and Anas Bedraoui. "CXAI: Explaining Convolutional Neural Networks for Medical Imaging Diagnostic". In: *Electronics* 11.11 (2022). ISSN: 2079-9292. DOI: 10.3390/electronics11111775. URL: https://www.mdpi.com/2079-9292/11/11/1775 (cit. on p. 20).

[14] Ibomoiye Domor Mienye, Theo G Swart, George Obaido, Matt Jordan, and Philip Ilono. "Deep convolutional neural networks in medical image analysis: A review". In: *Information* 16.3 (2025), p. 195 (cit. on p. 20).

[15] Akhil Jethwa. *Understanding Convolutional Hyperparameters*. https://www.kaggle.com/code/akhiljethwa/understanding-convolutional-hyperparameters. Accessed: 2025-05-11. 2023 (cit. on p. 20).

[16] Ahmad Waleed Salehi, Shakir Khan, Gaurav Gupta, Bayan Ibrahimm Alabduallah, Abrar Almjally, Hadeel Alsolai, Tamanna Siddiqui, and Adel Mellit. "A Study of CNN and Transfer Learning in Medical Imaging: Advantages, Challenges, Future Scope". In: *Sustainability* 15.7 (Mar. 29, 2023), p. 5930. ISSN: 2071-1050. DOI: 10.3390/su15075930. URL: https://www.mdpi.com/2071-1050/15/7/5930 (cit. on p. 22).

[17] Nima Tajbakhsh, Jae Y Shin, Suryakanth R Gurudu, R Todd Hurst, Christopher B Kendall, Michael B Gotway, and Jianming Liang. "Convolutional neural networks for medical image analysis: Full training or fine tuning?" In: *IEEE transactions on medical imaging* 35.5 (2016), pp. 1299–1312 (cit. on p. 22).

[18] Laith Alzubaidi, J. Santamaría, Mohamed Manoufali, Beadaa Mohammed, Mohammed A. Fadhel, Jinglan Zhang, Ali H. Al-Timemy, Omran Al-Shamma, and Ye Duan. *MedNet: Pre-trained Convolutional Neural Network Model for the Medical Imaging Tasks*. 2021. arXiv: 2110.06512 [cs.CV]. URL: https://arxiv.org/abs/2110.06512 (cit. on p. 22).

[19] Muhammad Maaz Irfan, Sheraz Ali, Irfan Yaqoob, and Numan Zafar. "Towards Deep Learning: A Review On Adversarial Attacks". In: *2021 International Conference on Artificial Intelligence (ICAI)*. 2021 International Conference on Artificial Intelligence

(ICAI). Apr. 2021, pp. 91–96. DOI: 10.1109/ICAI52203.2021.9445247. URL: https://ieeexplore.ieee.org/document/9445247/ (cit. on p. 23).

[20] Jing Lin, Long Dang, Mohamed Rahouti, and Kaiqi Xiong. *ML Attack Models: Adversarial Attacks and Data Poisoning Attacks*. Dec. 6, 2021. DOI: 10.48550/arXiv.2112.02797. arXiv: 2112.02797 [cs]. URL: http://arxiv.org/abs/2112.02797. Pre-published (cit. on p. 23).

[21] Rahul Paul, Matthew Schabath, Robert Gillies, Lawrence Hall, and Dmitry Goldgof. "Mitigating Adversarial Attacks on Medical Image Understanding Systems". In: *2020 IEEE 17th International Symposium on Biomedical Imaging (ISBI)*. 2020 IEEE 17th International Symposium on Biomedical Imaging (ISBI). Apr. 2020, pp. 1517–1521. DOI: 10.1109/ISBI45749.2020.9098740. URL: https://ieeexplore.ieee.org/document/9098740/ (cit. on p. 23).

[22] H. Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Agüera y Arcas. *Communication-Efficient Learning of Deep Networks from Decentralized Data*. Jan. 26, 2023. DOI: 10.48550/arXiv.1602.05629. arXiv: 1602.05629 [cs]. URL: http://arxiv.org/abs/1602.05629 (cit. on pp. 25, 27).

[23] Hao Guan, Pew-Thian Yap, Andrea Bozoki, and Mingxia Liu. "Federated Learning for Medical Image Analysis: A Survey". In: *Pattern Recognition* 151 (July 2024), p. 110424. ISSN: 00313203. DOI: 10.1016/j.patcog.2024.110424. URL: https://linkinghub.elsevier.com/retrieve/pii/S0031320324001754 (cit. on pp. 25, 27).

[24] Priyanka Mary Mammen. *Federated Learning: Opportunities and Challenges*. Jan. 14, 2021. DOI: 10.48550/arXiv.2101.05428. arXiv: 2101.05428 [cs]. URL: http://arxiv.org/abs/2101.05428 (cit. on pp. 26, 29).

[25] Viraaji Mothukuri, Reza M. Parizi, Seyedamin Pouriyeh, Yan Huang, Ali Dehghantanha, and Gautam Srivastava. "A Survey on Security and Privacy of Federated Learning". In: *Future Generation Computer Systems* 115 (Feb. 2021), pp. 619–640. ISSN: 0167739X. DOI: 10.1016/j.future.2020.10.007. URL: https://linkinghub.elsevier.com/retrieve/pii/S0167739X20329848 (cit. on pp. 26, 27, 29).

[26] Chen Zhang, Yu Xie, Hang Bai, Bin Yu, Weihong Li, and Yuan Gao. "A Survey on Federated Learning". In: *Knowledge-Based Systems* 216 (Mar. 2021), p. 106775. ISSN: 09507051. DOI: 10.1016/j.knosys.2021.106775. URL: https://linkinghub.elsevier.com/retrieve/pii/S0950705121000381 (cit. on p. 26).

[27] Naif Alkhunaizi, Dmitry Kamzolov, Martin Takáč, and Karthik Nandakumar. *Suppressing Poisoning Attacks on Federated Learning for Medical Imaging*. July 15, 2022. DOI: 10.48550/arXiv.2207.10804. arXiv: 2207.10804 [cs]. URL: http://arxiv.org/abs/2207.10804 (cit. on p. 27).

[28] Malhar S. Jere, Tyler Farnan, and Farinaz Koushanfar. "A Taxonomy of Attacks on Federated Learning". In: *IEEE Security & Privacy* 19.2 (Mar. 2021), pp. 20–28. ISSN: 1558-4046. DOI: 10.1109/MSEC.2020.3039941. URL: https://ieeexplore.ieee.org/document/9308910/ (cit. on p. 29).

[29] Tianyu Gu, Brendan Dolan-Gavitt, and Siddharth Garg. *BadNets: Identifying Vulnerabilities in the Machine Learning Model Supply Chain*. Mar. 11, 2019. DOI: 10.48550/arXiv.1708.06733. arXiv: 1708.06733 [cs]. URL: http://arxiv.org/abs/1708.06733. Pre-published (cit. on p. 31).

[30] Yiming Li, Yong Jiang, Zhifeng Li, and Shu-Tao Xia. "Backdoor Learning: A Survey". In: *IEEE Transactions on Neural Networks and Learning Systems* 35.1 (Jan. 2024), pp. 5–22. ISSN: 2162-2388. DOI: 10.1109/TNNLS.2022.3182979. URL: https://ieeexplore.ieee.org/document/9802938/ (cit. on p. 32).

[31] Ziteng Sun, Peter Kairouz, Ananda Theertha Suresh, and H. Brendan McMahan. *Can You Really Backdoor Federated Learning?* Dec. 2, 2019. DOI: 10.48550/arXiv.1911.07963. arXiv: 1911.07963 [cs]. URL: http://arxiv.org/abs/1911.07963 (cit. on p. 32).

[32] Wei Guo, Benedetta Tondi, and Mauro Barni. "An Overview of Backdoor Attacks Against Deep Neural Networks and Possible Defences". In: *IEEE Open Journal of Signal Processing* 3 (2022), pp. 261–287. ISSN: 2644-1322. DOI: 10.1109/OJSP.2022.3190213. URL: https://ieeexplore.ieee.org/document/9827581/ (cit. on p. 34).

[33] Edward Chou, Florian Tramèr, and Giancarlo Pellegrino. "SentiNet: Detecting Localized Universal Attacks Against Deep Learning Systems". In: *2020 IEEE Security and Privacy Workshops (SPW)*. 2020, pp. 48–54. DOI: 10.1109/SPW50608.2020.00025 (cit. on p. 34).

[34] Yansong Gao, Change Xu, Derui Wang, Shiping Chen, Damith C. Ranasinghe, and Surya Nepal. "STRIP: a defence against trojan attacks on deep neural networks". In: *Proceedings of the 35th Annual Computer Security Applications Conference*. ACSAC '19. San Juan, Puerto Rico, USA: Association for Computing Machinery, 2019, pp. 113–125. ISBN: 9781450376280. DOI: 10.1145/3359789.3359790. URL: https://doi.org/10.1145/3359789.3359790 (cit. on p. 34).

[35] Xueluan Gong, Yanjiao Chen, Qian Wang, and Weihan Kong. "Backdoor Attacks and Defenses in Federated Learning: State-of-the-Art, Taxonomy, and Future Directions". In: *IEEE Wireless Communications* 30.2 (2023), pp. 114–121. DOI: 10.1109/MWC.017.2100714 (cit. on p. 35).

[36] Suyi Li, Yong Cheng, Wei Wang, Yang Liu, and Tianjian Chen. *Learning to Detect Malicious Clients for Robust Federated Learning*. 2020. arXiv: 2002.00211 [cs.LG]. URL: https://arxiv.org/abs/2002.00211 (cit. on p. 35).

[37] Sebastien Andreina, Giorgia Azzurra Marson, Helen Möllering, and Ghassan Karame. "BaFFLe: Backdoor Detection via Feedback-based Federated Learning". In: *2021 IEEE 41st International Conference on Distributed Computing Systems (ICDCS)*. 2021, pp. 852–863. DOI: 10.1109/ICDCS51616.2021.00086 (cit. on p. 35).

[38] Chen Wu, Xian Yang, Sencun Zhu, and Prasenjit Mitra. *Mitigating Backdoor Attacks in Federated Learning*. 2021. arXiv: 2011.01767 [cs.CR]. URL: https://arxiv.org/abs/2011.01767 (cit. on p. 35).

[39] Panagiotis V. Nomikos and Ioannis S. Antoniadis. “Introduction to Brain Tumors”. In: *Imaging in Clinical Oncology*. Ed. by Athanasios D. Gouliamos, John A. Andreou, and Paris A. Kosmidis. Cham: Springer International Publishing, 2018, pp. 131–134. ISBN: 978-3-319-68872-5 978-3-319-68873-2. DOI: 10.1007/978-3-319-68873-2_16. URL: http://link.springer.com/10.1007/978-3-319-68873-2_16 (cit. on p. 36).

[40] Evgin Goceri. “Medical image data augmentation: techniques, comparisons and interpretations”. In: *Artificial Intelligence Review* 56.11 (2023), pp. 12561–12605 (cit. on p. 37).

[41] Manika Jha, Richa Gupta, and Rajiv Saxena. “A framework for in-vivo human brain tumor detection using image augmentation and hybrid features”. In: *Health Information Science and Systems* 10.1 (2022), p. 23 (cit. on p. 38).

[42] Karen Simonyan and Andrew Zisserman. “Very deep convolutional networks for large-scale image recognition”. In: *arXiv preprint arXiv:1409.1556* (2014) (cit. on p. 46).

[43] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. “Mobilenetv2: Inverted residuals and linear bottlenecks”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2018, pp. 4510–4520 (cit. on p. 46).

[44] Yuki Matsuo and Kazuhiro Takemoto. “Backdoor Attacks to Deep Neural Network-Based System for COVID-19 Detection from Chest X-ray Images”. In: *Applied Sciences* 11.20 (20 Jan. 2021), p. 9556. ISSN: 2076-3417. DOI: 10.3390/app11209556. URL: https://www.mdpi.com/2076-3417/11/20/9556 (cit. on p. 47).

[45] Ramprasaath R. Selvaraju, Michael Cogswell, Abhishek Das, Ramakrishna Vedantam, Devi Parikh, and Dhruv Batra. “Grad-CAM: Visual Explanations from Deep Networks via Gradient-based Localization”. In: *International Journal of Computer Vision* 128.2 (Feb. 2020), pp. 336–359. ISSN: 0920-5691, 1573-1405. DOI: 10.1007/s11263-019-01228-7. arXiv: 1610.02391 [cs]. URL: http://arxiv.org/abs/1610.02391 (cit. on p. 52).

[46] Mohamed Musthafa M, Mahesh T. R, Vinoth Kumar V, and Suresh Guluwadi. “Enhancing Brain Tumor Detection in MRI Images through Explainable AI Using Grad-CAM with Resnet 50”. In: *BMC Medical Imaging* 24.1 (May 11, 2024), p. 107. ISSN: 1471-2342. DOI: 10.1186/s12880-024-01292-7. URL: https://doi.org/10.1186/s12880-024-01292-7 (cit. on p. 52).

[47] Munachiso Nwadike, Takumi Miyawaki, Esha Sarkar, Michail Maniatakos, and Farah Shamout. *Explainability Matters: Backdoor Attacks on Medical Imaging*. Dec. 30, 2020. DOI: 10.48550/arXiv.2101.00008. arXiv: 2101.00008 [cs]. URL: http://arxiv.org/abs/2101.00008. Pre-published (cit. on p. 52).

[48] Hong-You Chen, Cheng-Hao Tu, Ziwei Li, Han-Wei Shen, and Wei-Lun Chao. “On the importance and applicability of pre-training for federated learning”. In: *arXiv preprint arXiv:2206.11488* (2022) (cit. on p. 60).

Dedications

A mio padre, la mia più grande fonte di ispirazione, che mi ha insegnato a non avere eroi, diventando il mio.

A mia madre e a mia nonna, che riescono sempre a gioire più di me per le mie vittorie.

Alla mia fan numero uno, che lo scorso anno ha tenuto a ricordarmi ciò che conta davvero nella vita.

A Bebo, il mio coniglio terapista.

And a final reminder to my future self and to everyone who will read this thesis:
"You are who you choose to be"
- The Iron Giant (1999), Brad Bird