MULTI-MODEL DEEP LEARNING FOR BREAST CANCER: A COMPARATIVE ANALYSIS

by
Kağan Tek

Engineering Project Report

Yeditepe University
Faculty of Engineering
Department of Computer Engineering
2025

MULTI-MODEL DEEP LEARNING FOR BREAST CANCER: A COMPARATIVE ANALYSIS

APPROVED BY:

Assoc. Prof. Dr. Dionysis Goularas (Supervisor) . . . . . .

Asst. Prof. Dr. Mustafa Bülent Mutluoğlu . . . . . .

Prof. Dr. Emin Erkan Korkmaz . . . . . .

DATE OF APPROVAL: .../.../2025

ACKNOWLEDGEMENTS

First of all I would like to thank my advisor Assist. Prof. Dr. Dionysis Goularas for his guidance and support throughout my project.

Also I would like to thank my parents for their support and encouragement throughout my education up to the present.

ABSTRACT

MULTI-MODEL DEEP LEARNING FOR BREAST CANCER: A COMPARATIVE ANALYSIS

Breast cancer remains a leading cause of mortality worldwide, underscoring the critical need for early and accurate detection. Advancements in imaging modalities such as mammography, ultrasound, and magnetic resonance imaging (MRI) have improved diagnostic capacities as well as histopathology practices for classifying cancer. This study proposes a comparative analysis of CNN models trained in different datasets of medical imaging modality to observe the performance of dedicated models for the classification of breast cancer. Each model's performance will be evaluated using certain metrics such as sensitivity, specificity, precision, f1 score, etc. The deep learning models will be deployed on a web app, open for testing and utilization. The project emphasizes the ethical implementation of AI, incorporating bias mitigation through data diversity. The results will provide actionable insights to compare the types of medical imaging techniques and model architectures, potentially discovering the best alternative to improve early detection rates while reducing diagnostic costs and variability.

ÖZET

MEME KANSERİ SINIFLANDIRMASI İÇİN ÇOKLU MODEL DERİN ÖĞRENME YAKLAŞIMI: KARŞILAŞTIRMALI BİR İNCELEME

Meme kanseri, dünya genelinde en yaygın ölüm nedenlerinden biri olmaya devam etmekte olup, erken ve doğru tanının önemini her geçen gün vurgulamaktadır. Mamografi, ultrason ve manyetik rezonans görüntüleme (MRG) gibi görüntüleme yöntemlerinde kaydedilen ilerlemeler, tanısal doğruluğun artırılmasının yanı sıra, kanserin sınıflandırılmasına yönelik histopatolojik uygulamalar da tanı konulmasına yönelik opsiyonları yaygın kılmaktadır. Bu çalışmada, farklı tıbbi görüntüleme modalitelerine ait veri kümeleri üzerinde eğitilmiş konvolüsyonel sinir ağı (CNN) modellerinin karşılaştırmalı analizi gerçekleştirilerek, meme kanseri sınıflandırmasına yönelik özel modellerin performansları değerlendirilecektir. Her bir modelin başarımı; duyarlılık (sensitivity), spesifite (specificity), kesinlik (precision), F1 skoru gibi çeşitli değerlendirme metrikleri kullanılarak ölçülecektir. Elde edilen derin öğrenme modelleri, kullanıcıların erişimine açık bir web uygulaması üzerinden test edilip kullanılabilecek şekilde dağıtılacaktır. Proje kapsamında, veri çeşitliliği aracılığıyla önyargının azaltılması sağlanarak yapay zekânın etik kullanımına da özel önem verilmektedir. Çalışmanın sonuçlarının, farklı tıbbi görüntüleme teknikleri ve model mimarileri arasındaki karşılaştırmalara yönelik uygulanabilir bulgular sunması ve böylece erken tanı oranlarını artırırken tanı maliyetlerini ve değişkenliklerini azaltabilecek en etkili alternatiflerin belirlenmesine katkıda bulunması beklenmektedir.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF SYMBOLS/ABBREVIATIONS

AI	Artificial Intelligence
AUC	Area Under the Curve
ANN	Artificial Neural Networks
CAD	Computer Aided Diagnosis
CNN	Convolutional Neural Network
DL	Deep Learning
GPU	Graphics Processing Unit
IDC	Invasive Ductal Carcinoma
MGH	Massachusetts General Hospital
MIT	Massachusetts Institute of Technology
SGD	Stochastic Gradient Descent
TPU	Tensor Processing Unit

1. INTRODUCTION

This section introduces the project, problem definition, requirements and the basic knowledge which is deeply relational with the project such as deep learning, deep learning in the medical field and CNNs.

1.1. Basic Knowledge

1.1.1. Deep Learning

Deep learning is a sub-field of machine learning that studies artificial neural networks with many hierarchical layers. Although the basic back-propagation algorithm dates to the mid-1980s, recent leaps in labeled data availability, GPU/TPU acceleration, new optimization tricks (e.g., batch-normalization, residual connections) and open-source frameworks have turned deep nets into state-of-the-art engines for perception, language, and scientific discovery (LeCun et al., 2015)[1]. What follows summarizes the lineage of these ideas, the core architectural families, and why they matter for computer-aided breast cancer diagnosis.

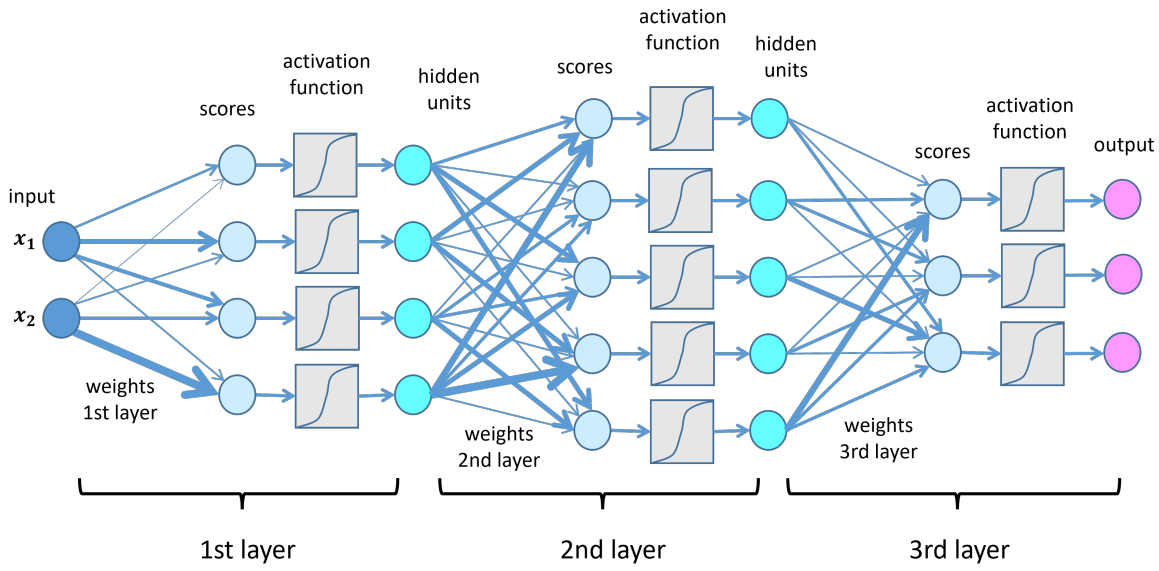

Figure 1.1. An Example of a Deep Learning Neural Network Architecture

Deep learning models are differentiable function approximators: they map an input x to an output y through a stack of parametrized linear transforms and nonlinearities. Training amounts to minimizing a loss with SGD and backpropagation, first laid out by Rumelhart, Hinton, & Williams in 1986 (Rumelhart et al., 1986)[2]. Because each hidden layer learns a progressively more abstract representation, deep nets can, in principle, disentangle complex structure from raw signals: images, sound waves, genomic sequences, etc.

A 2017 survey covering more than 300 papers concluded that CNN-based systems match, or exceed expert radiologist performance on tasks such as tumour detection, organ segmentation and disease grading (Litjens et al., 2017)[3]. Subsequent work on digital pathology has shown that residual and densely connected nets can identify IDC patches with AUCs > 0.90 (Celik et al., 2020)[4], underscoring the promise of deep feature hierarchies for breast-cancer screening.

1.1.2. Deep Learning in the Medical Field

Over the past decade, DL has moved from academic curiosity to a clinically relevant technology stack, powering tools that already equal, or surpass specialists for narrowly defined tasks. Landmark convolutional-network studies showed dermatologist-level skin-lesion classification (Esteva et al., 2017)[5] and radiologist-level pneumonia detection (Rajpurkar et al., 2017)[6] on chest X-rays, convincing many clinicians that data-driven pattern recognition could handle complex image interpretation. Soon after, fully three-dimensional CNNs delivered state-of-the-art performance in low-dose CT lung-cancer screening, while weakly supervised pipelines pushed whole-slide pathology into “clinical-grade” territory with million-tile gigapixel slides.

Within this landscape, the application of deep learning to breast cancer detection has become a particularly active and promising area of research (Shen et al., 2019)[7]. The diagnostic pathway for breast cancer often involves multiple imaging modalities, each with unique characteristics that present distinct challenges and opportunities for AI. For mammography, CNNs are trained to identify subtle signs of malignancy, such as suspicious microcalcification clusters and spiculated mass margins, which can be difficult to discern in dense breast tissue. In ultrasound imaging, deep learning models excel at differentiating between benign cysts and potentially malignant solid masses, a critical step in reducing unnecessary biopsies. For histopathology, models analyze gigapixel whole-slide images to automatically identify cancerous regions and even predict tumor grade, a task that is both time-consuming and subject to inter-observer variability among pathologists. The success of DL across these diverse

tasks has demonstrated its potential to create a more accurate, efficient, and standardized diagnostic workflow, motivating the central goal of this project: to perform a comparative analysis and identify the optimal CNN architectures for each of these critical imaging modalities.

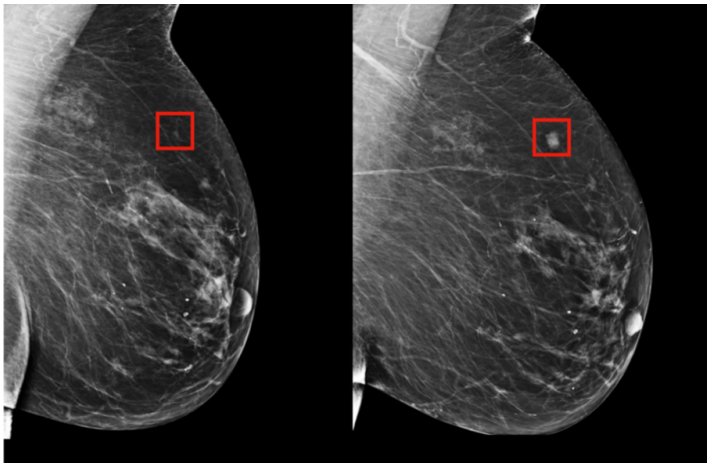

Figure 1.2. MIT/MGH's DL Model Predicting Breast Cancer up to Five Years In Advance (Portnoi et al., 2019) [8]

1.1.3. Artificial Neural Networks

At the heart of deep learning lies the Artificial Neural Network (ANN), a computational model inspired by the interconnected structure of neurons in the human brain (McCulloch et al., 1943)[9]. The most fundamental form of this model consists of an input layer, one or more hidden layers, and an output layer. Each 'neuron' or node in a layer receives inputs from the nodes in the preceding layer. These inputs are multiplied by learnable 'weights,' which signify the strength of their connection. The neuron then sums these weighted inputs, adds a bias term, and passes the result through a non-linear activation function (such as Sigmoid or ReLU) (Nair et al., 2010)[10]. This activation function is critical; it introduces non-linearity into the system, allowing the network to learn and model complex, real-world relationships that go far beyond simple linear correlations. Through the training process of backpropagation, the network systematically adjusts these weights and biases to minimize the difference between its predictions and the actual ground truth, effectively learning from the data. While this foundational architecture is powerful for a variety of tasks, its simple structure treats all inputs independently, making it less than ideal for data with inherent spatial structure, like

images. This limitation directly motivated the development of more specialized architectures, such as the Convolutional Neural Networks (CNNs) discussed next.

1.1.4. Convolutional Neural Networks

Convolutional Neural Networks (CNNs) represent a specialized class of deep neural networks that have become the de-facto standard for tasks involving grid-like data, most notably image analysis. Their design is loosely inspired by the organization of the animal visual cortex, where individual neurons respond to stimuli only in a restricted region of the visual field known as the Receptive Field. The core operation of a CNN is the 'convolution,' where a small filter, or kernel, slides across the input image to produce a feature map. Each filter is designed to detect a specific low-level feature, such as an edge, a corner, or a particular color gradient (LeCun et al., 1998)[11]. This approach has two key advantages: parameter sharing, where the same filter is used across the entire image, drastically reducing the number of parameters compared to a standard neural network, and translation equivariance, meaning the network can detect a feature regardless of where it appears in the image.

A typical CNN architecture consists of a sequence of layers. After an initial convolutional layer extracts basic features, a pooling layer (e.g., max pooling) is often applied to downsample the feature map. This reduces the computational load and makes the detected features more robust to minor shifts in their position. By stacking multiple convolutional and pooling layers, the network learns a rich spatial hierarchy of features: early layers learn simple shapes, which are then combined by deeper layers to form more complex, abstract concepts like textures, patterns, and object parts (Krizhevsky et al., 2017)[12]. Finally, after a sufficient hierarchy of features has been learned, the high-level feature maps are 'flattened' into a one-dimensional vector and passed to a standard fully connected neural network for the final classification task. This powerful, hierarchical structure is what allows CNNs to achieve state-of-the-art results in image recognition and is the foundational technology for the more advanced architectures explored in this paper.

1.2. Problem Definition

Breast cancer represents a significant global health crisis, standing as one of the most commonly diagnosed cancers and a leading cause of cancer-related mortality among women worldwide (Bray et al., 2024)[13]. The cornerstone of improving patient outcomes and survival rates is early and accurate diagnosis. The current clinical standard for diagnosis relies

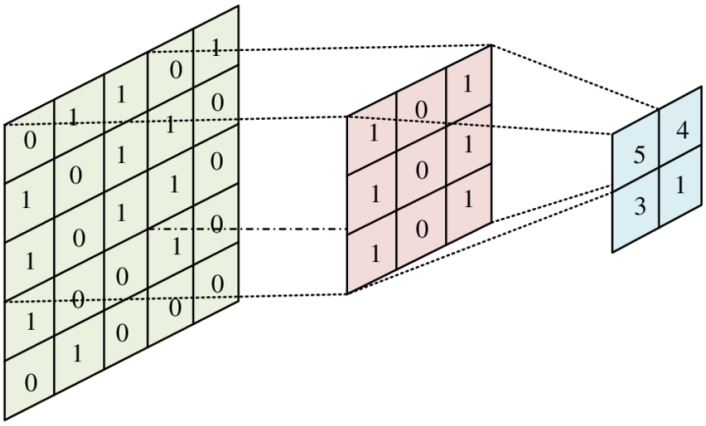

Figure 1.3. Example of a Convolution Operation

on the expert interpretation of various medical imaging modalities, a process that is highly demanding and subject to inherent human limitations. Radiologists and pathologists face challenges such as high caseloads leading to fatigue, subjective interpretation which can result in inter-observer variability, and the difficulty of detecting subtle abnormalities in complex images. These factors can contribute to diagnostic delays or errors, underscoring the urgent need for robust, reliable, and efficient tools to augment the capabilities of clinical specialists.

While deep learning has emerged as a powerful solution to these challenges, the effective application of this technology introduces a new, more specific technical problem. The primary imaging modalities used in breast cancer diagnosis, mammography, ultrasound, and histopathology produce visually distinct data that capture different biological characteristics (Iranmakani et al., 2020)[14]. A mammogram requires the analysis of macro-structures and tissue densities, whereas a histopathology slide demands fine-grained analysis of cellular morphology and texture. Consequently, a Convolutional Neural Network (CNN) architecture that excels at one task is not guaranteed to be optimal for another. Top-performing architectures like ResNet50, VGG16, and DenseNet121 each possess unique design principles such as residual connections, depth, or feature reuse that may make them better suited for some data types over others.

This leads to the central problem this project aims to solve: there is a lack of systematic, evidence-based guidance for selecting the most effective CNN architecture for each specific breast imaging modality. Without a direct comparative analysis, the choice of a model becomes speculative. Therefore, this project addresses this knowledge gap by performing a rigorous, head-to-head evaluation of these three prominent CNNs across all three imaging modalities to definitively identify which architectural approach yields the best diagnostic

performance for each specific use case.

1.2.1. Motivation

The primary motivation for this project stems from the critical need to enhance the reliability and effectiveness of CAD systems for breast cancer. While the potential of deep learning is undisputed, its practical implementation is hindered by a crucial uncertainty (Jiménez-Gaona et al., 2020)[15]: the performance of a given CNN architecture is deeply intertwined with the specific characteristics of the imaging modality it is applied to (Luo et al., 2024)[16]. A model architecture that is optimal for identifying microcalcifications in a mammogram may not be the best choice for classifying cellular structures in a high-resolution histopathology slide. This variability necessitates a move beyond generalized approaches toward a more specialized, evidence-based strategy for model selection.

Therefore, the central goal of this project is to provide a clear, rigorous, and actionable answer to the question: Which deep learning architecture is best suited for each specific breast imaging modality? To achieve this, this study is motivated to perform a direct comparative analysis of three influential CNN architectures, ResNet50, VGG16, and DenseNet121 across mammography, ultrasound, and histopathology datasets. By systematically evaluating their performance, this project aims to generate valuable insights that can guide the development of more accurate and modality-aware diagnostic tools. Furthermore, to demonstrate the practical application of these findings, a secondary goal is to develop and deploy a functional web-based portal that serves these trained models, bridging the gap between theoretical research and a tangible, proof-of-concept clinical support system.

Furthermore, the diagnostic process is not only complex but also time-intensive. The time of a skilled radiologist or pathologist is a finite and valuable resource. In a healthcare system facing ever-increasing demand, lengthy interpretation times for a single case can create bottlenecks, extending waiting periods for other patients. Because breast cancer is a time-critical condition where early detection directly influences survival rates, easing and accelerating the diagnostic workflow is paramount. By developing highly accurate automated tools, we can augment the capabilities of medical professionals, allowing them to triage cases more efficiently and focus their expertise on the most challenging diagnoses.

1.3. Requirements

1.3.1. Functional Requirements

1.3.1.1. Modality and Model Selection.

- The system shall provide distinct user interfaces or pages for each of the three imaging modalities: Mammography, Ultrasound, and Histopathology.
- Within each modality page, there must be 6 trained models under two categories of training principles, the first one being initially unfrozen and the second one being fine-tuned which will be explained in the analysis and design section.
- The user must be able to select which of the three trained models (ResNet50, VGG16, or DenseNet121) to use for the prediction.

1.3.1.2. Image Upload and Handling.

- The system must provide an interface for the user to upload a single image file.
- The system shall accept common image formats, including .png, .jpg, and .jpeg. Upon upload, the system shall securely handle the image file for processing.

1.3.1.3. Prediction Generation.

- The system shall feed the preprocessed image into the user-selected, pre-trained CNN model.
- The model shall generate a binary classification prediction (e.g., ”Tumor” vs. ”No Tumor”, ”Malignant” vs. ”Benign”).
- The system shall also output a quantitative confidence score or probability associated with the prediction.
- There must also be an additional page to make simultaneous prediction for models that are considered in the same cohort for efficient comparison.

1.3.1.4. Results Display.

- The system must display the final classification label (e.g., ”Tumor Detected”) and its confidence score to the user.
- The web application must also feature dedicated pages or sections to display the historical performance metrics (Accuracy, Precision, Recall, F1-Score) and visualizations (e.g., Confusion Matrix, ROC Curve) for each of the eighteen trained models.
- The user must be able to view the training notebooks for the models.

1.3.2. Non-Functional Requirements

1.3.2.1. Performance.

- Loaded models must be cached in order to not get loaded each time, reducing computation cost.
- The trained models must achieve a high level of diagnostic accuracy, with an F1-score of at least 0.70 on their respective hold-out test sets.

1.3.2.2. Usability.

- The web interface shall be intuitive, with clear instructions and navigation, allowing a non-technical user to easily upload an image and obtain a prediction.
- Error handling shall be implemented to gracefully manage invalid file uploads (e.g., non-image files).

1.3.2.3. Reliability.

- The web application shall be stable and not crash during standard user operations (image upload, prediction).
- The prediction models must load successfully without errors upon application startup.

1.3.2.4. Security.

- While this is a proof-of-concept system, it should follow basic security principles.
- Uploaded user images should be treated as temporary files and should be deleted from the server after the prediction is made and displayed.
- Files which are not images must not be permitted to get uploaded.

1.3.2.5. Scalability.

- The system architecture (specifically the Flask application) should be structured in a modular way that would allow for the easy addition of new models or imaging modalities in the future without requiring a complete redesign.

1.3.2.6. Reusability.

- Components will be developed with modularity in mind, ensuring they can be easily adapted and repurposed for different sections of this project or future related initiatives.

2. RELATED WORK

This chapter reviews the existing body of research relevant to this project. It begins by tracing the development of the core technology, Convolutional Neural Networks (CNNs), focusing on the key architectural milestones that led to the models used in this study. It then explores the application of these models in the specific domains of mammography, ultrasound, and histopathology for breast cancer diagnosis, establishing the context and identifying the research gap that this comparative analysis aims to fill.

2.1. Convolutional Neural Networks

The journey of the modern Convolutional Neural Network began long before the recent deep learning explosion. Early pioneering work by LeCun which is cited above introduced the LeNet-5 architecture, which successfully applied the core principles of convolutions and pooling layers to handwritten digit recognition. However, the true potential of CNNs was not fully realized until the advent of large-scale datasets and powerful GPU hardware. The turning point came in 2012 when Krizhevsky et al. introduced AlexNet, a much deeper and wider CNN that won the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) by a significant margin. AlexNet's success demonstrated that deep CNNs, trained on sufficient data, could achieve state-of-the-art performance in complex image recognition tasks, sparking the deep learning revolution. This breakthrough paved the way for the development of even more sophisticated architectures, including the three key models evaluated in this project.

VGG16: The VGG16 architecture, introduced by Simonyan and Zisserman from the Visual Geometry Group at the University of Oxford (Simonyan etl a., 2014)[17], was a landmark model that achieved top results in the ILSVRC 2014 challenge. Its core design philosophy was a direct investigation into the effect of network depth on performance, guided by the principles of simplicity and uniformity. The key contribution of the VGG models was the compelling demonstration that state-of-the-art performance could be achieved not through complex, specialized layers, but through the brute-force stacking of a single, simple, repeated building block.

The fundamental building block of VGG16 consists of a sequence of 3x3 convolutional layers followed by a 2x2 max-pooling layer. The choice of a 3x3 kernel is the smallest size that can still capture the concepts of left-right, up-down, and center. The genius of the

VGG design lies in stacking these small filters. For instance, a stack of two consecutive 3x3 convolutional layers has an effective receptive field of 5x5; a stack of three has a receptive field of 7x7. This approach has two major advantages over using a single larger filter. First, it incorporates more non-linear Rectified Linear Unit (ReLU) activation functions between the layers, making the network's decision function more discriminative. Second, it reduces the total number of parameters. For example, a stack of three 3x3 layers involves 3 * (3*3) = 27 weights, whereas a single 7x7 layer requires 7*7 = 49 weights—a reduction of nearly 45

The overall VGG16 architecture is a sequence of five of these convolutional blocks. As the input image passes through the network, the spatial dimensions are progressively reduced by the max-pooling layers (from 224x224 down to 7x7), while the depth (the number of channels or filters) is systematically increased, typically doubling after each pooling layer (from 64 to 128, to 256, and finally to 512). This design is based on the idea that as spatial information is compressed, the network must increase the number of feature maps to capture more complex and abstract visual patterns.

After the final convolutional block, the resulting feature maps are flattened into a one-dimensional vector and fed into a classifier head consisting of three fully-connected layers. While the convolutional base is highly effective, this classifier head is the architecture's primary drawback. The first two fully-connected layers contain 4096 neurons each, and the final layer contains 1000 neurons (for the 1000 ImageNet classes). These three layers alone account for the vast majority of the model's 138 million parameters, making VGG16 very large, memory-intensive, and computationally expensive. Despite this, its elegant and powerful structure established a new standard for network design and remains a popular baseline and feature extractor for many computer vision tasks today.

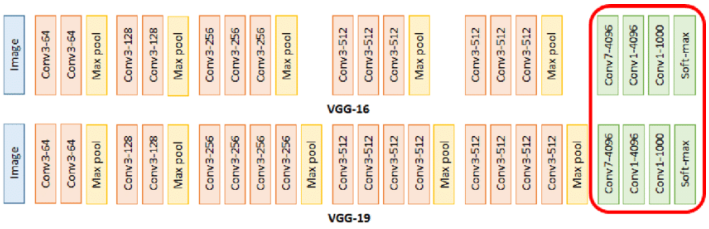

Figure 2.1. VGG16 and VGG19's CNN Architecture

ResNet50: As network architectures following the VGG model grew deeper, a significant obstacle emerged known as the degradation problem. Researchers observed that simply stacking more layers onto an already deep model paradoxically led to higher training and test

error. This was not a case of overfitting, but rather an optimization challenge: very deep ”plain” networks struggled to learn even simple identity functions, meaning that adding a new layer could actually hurt performance if that layer could not at least learn to pass its input through unchanged. The Residual Network (ResNet) (He et al., 2015)[18] in their groundbreaking paper, presented a revolutionary and remarkably simple solution to this degradation problem, enabling the stable training of networks that were substantially deeper than anything previously conceived.

The core architectural innovation of ResNet is the residual block, which features a ”skip connection” or ”shortcut.” In a traditional network, a block of layers attempts to learn a direct mapping, H(x), from an input x. In a residual block, the layers are instead tasked with learning a residual mapping, F(x). The output of the block then becomes H(x) = F(x) + x, where the original input x is added back via the skip connection. The critical insight behind this design is that it is far easier for a network to learn to push the weights of the residual function F(x) towards zero than it is to learn the identity mapping H(x) = x from scratch. If an identity mapping is optimal for a given layer, the network can easily achieve it. This structure creates a direct, unimpeded path for the gradient to flow during backpropagation, effectively combating the vanishing gradient problem that plagued earlier deep models and allowing for stable training at extreme depths.

The ResNet50 variant, one of the most popular in the family, is a 50-layer deep network that employs an efficient ”bottleneck” design for its residual blocks to manage computational cost. Instead of using two 3x3 convolutions as in simpler ResNets, these bottleneck blocks use a sequence of 1x1, 3x3, and 1x1 convolutions. The first 1x1 convolution acts as a channel-wise dimensionality reduction, reducing the number of feature maps (the ”bottleneck”). This allows the computationally expensive 3x3 convolution to operate on a much smaller tensor. The final 1x1 convolution then restores the original channel depth. This design allows for a significant increase in network depth without a proportional increase in computational complexity. As a result, ResNet50 is not only more powerful than VGG16 but also vastly more efficient, with only 25 million parameters compared to VGG16’s 138 million. This combination of depth, innovative residual learning, and computational efficiency has made ResNet a foundational architecture in modern deep learning.

DenseNet121: The Densely Connected Convolutional Network (DenseNet) (Huang et al., 2018)[19], offers a compelling architectural evolution that builds on the concept of shortcut connections to maximize information flow. While ResNet improved gradient flow by

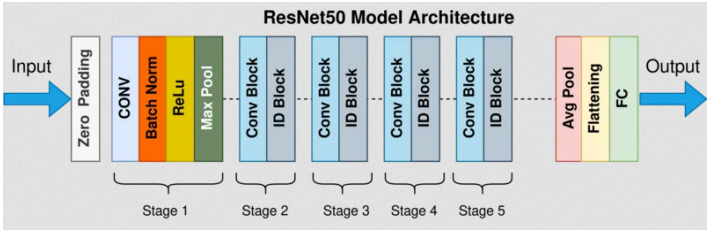

Figure 2.2. ResNet50's CNN Architecture

creating identity paths through summation, DenseNet proposed a more radical idea: what if every layer received direct input from all preceding layers? This is achieved through a simple yet powerful change in connectivity: instead of adding features from a skip connection, DenseNet concatenates feature maps.

The core of the architecture is the dense block. Within a dense block containing L layers, the input to the l-th layer is the concatenation of the feature maps from all L-1 previous layers. Its own output feature map is then passed on as an input to all subsequent L-l layers. This creates a dense web of L(L+1)/2 connections where each layer receives a "collective knowledge" from all layers before it. This design philosophy yields several profound advantages. First, it encourages extensive feature reuse, as features learned in early layers (e.g., edges) are directly available to much deeper layers, preventing the network from having to relearn redundant information. Second, it dramatically improves the flow of information and gradients throughout the network, a phenomenon the authors call "deep supervision," as each layer receives a more direct supervisory signal from the loss function.

To prevent the feature maps from growing infinitely large due to continuous concatenation, DenseNet introduces a hyperparameter called the growth rate (k). This parameter controls how many new feature maps each layer adds to the collective knowledge (e.g., k=32). This allows the network to be very deep while remaining surprisingly narrow and computationally lean. To handle the necessary down-sampling between stages, the network is divided into multiple dense blocks separated by transition layers. These transition layers consist of a 1x1 convolution to reduce the number of channels, followed by a 2x2 average pooling layer to reduce the spatial dimensions. The DenseNet121 variant used in this study is 121 layers deep, yet due to its novel design, it is extraordinarily parameter-efficient, containing only 8 million parameters. This is a small fraction of both ResNet50 (25M) and VGG16 (138M), making it a powerful and compelling architecture, especially for tasks where parameter efficiency and strong feature propagation are desirable.

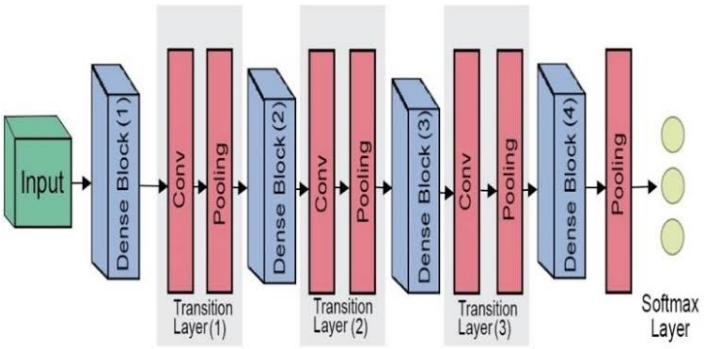

Figure 2.3. DenseNet121's CNN Architecture

2.2. Imaging Modalities in Breast Cancer Diagnosis

The clinical pathway for diagnosing breast cancer is a multi-stage, multi-modal process that relies on different imaging techniques to build a comprehensive picture of a patient's condition. Each modality provides unique insights by visualizing different properties of breast tissue, and each presents a distinct set of challenges for both human interpreters and artificial intelligence systems. The process typically begins with screening, moves to targeted diagnostic investigation, and culminates in a definitive pathological examination. Understanding the role and inherent complexities of each imaging type is fundamental to appreciating why a single deep learning architecture may not be universally optimal and why a comparative, modality-specific approach is necessary. This project focuses on the three primary modalities in this pathway: mammography, ultrasound, and histopathology.

Mammography: Mammography is a low-dose X-ray imaging technique that serves as the international gold standard for breast cancer screening, with also certain suspicions and concerns surrounding it (Naik et al., 2024)[20]. Its primary goal is the early detection of breast cancer in asymptomatic women, with the aim of identifying tumors before they become palpable or cause symptoms. The resulting grayscale images, or mammograms, reveal the internal structure of the breast, allowing radiologists to identify various potential indicators of malignancy. These include masses, which are analyzed based on their shape, size, and margin characteristics (e.g., a spiculated or "star-shaped" margin is highly suspicious); architectural distortions, where the normal tissue structure is disrupted without a clear mass; and, most critically, microcalcifications. These are tiny deposits of calcium that can appear as small, bright specks on a mammogram. While often benign, the morphology and distribution of these calcifications can be a key early sign of Ductal Carcinoma in Situ (DCIS), a

non-invasive form of breast cancer.

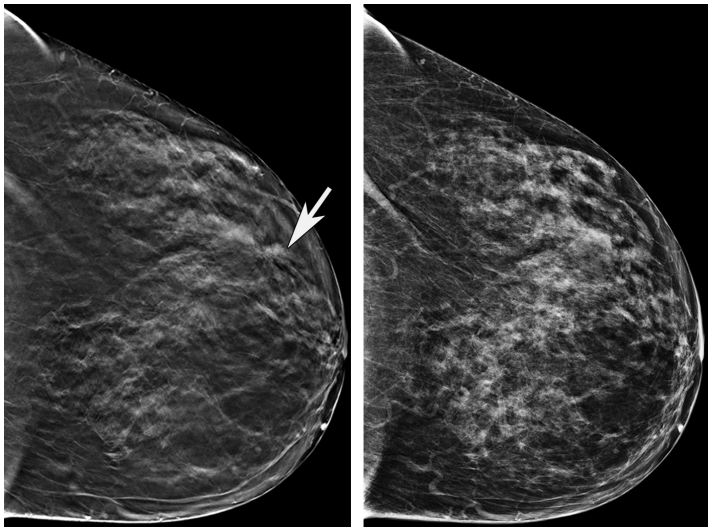

Figure 2.4. Example Mammmography Image

For a Convolutional Neural Network, analyzing mammograms presents several profound challenges. The first is the issue of low contrast and subtle features. Malignant lesions can have very similar pixel intensities to the surrounding glandular and fibrous tissue, making them difficult to distinguish. The second, and perhaps most significant, challenge is breast density (Payne et al., 2025)[21]. In patients with dense breasts, the large amount of fibroglandular tissue appears white on the mammogram, the same color as a tumor, creating a ”masking” effect that can obscure cancers and lower the sensitivity of the screening. A successful deep learning model for mammography must therefore be exceptionally adept at learning the subtle textural and morphological differences that define malignancy. It needs to function not just as a blob detector, but as a sophisticated pattern recognition system capable of identifying faint spicules on a mass margin or analyzing the heterogeneous shape of individual microcalcifications within a cluster, all while navigating the confounding background of varying tissue densities.

Ultrasound: Breast ultrasound is a non-invasive, radiation-free imaging technique that uses high-frequency sound waves to visualize breast tissue in real-time. Unlike mammography, it is not typically used for primary screening but serves as an indispensable diagnostic tool, often called the ”diagnostic workhorse” of breast imaging. Its most common clinical application is to further investigate abnormalities detected on a mammogram—especially in cases of dense breast tissue where mammography is less effective—or to evaluate a palpable

lump found during a physical exam. The primary strength of ultrasound lies in its exceptional ability to differentiate between solid masses, which may be cancerous, and simple fluid-filled cysts, which are almost always benign. This capability is critical for reducing the rate of unnecessary biopsies and providing immediate reassurance to patients with benign findings (Sehgal et al., 2006)[21].

From a deep learning perspective, the analysis of ultrasound images is fundamentally a task of shape and texture classification, but one that is fraught with unique challenges. A CNN model must learn to interpret key diagnostic features based on the BI-RADS (Breast Imaging Reporting and Data System) lexicon, such as the lesion's shape (e.g., oval vs. irregular), orientation (parallel vs. not parallel to the skin), margin characteristics (circumscribed vs. spiculated), and internal echo pattern (e.g., anechoic, hypoechoic, or heterogeneous). However, this task is complicated by several inherent properties of ultrasound imaging. The foremost is the presence of speckle noise, a granular pattern that is intrinsic to the coherent interference of sound waves and can degrade image quality. This noise can obscure the true margins of a lesion, making the crucial task of margin analysis difficult for a model. Furthermore, ultrasound is highly operator-dependent; the quality of the image and the visibility of the lesion can vary significantly based on the skill of the sonographer, the type of transducer used, and the pressure applied. A successful CNN for breast ultrasound must therefore be robust to this noise and variability, capable of learning key morphological features that are consistent and generalizable across different image qualities and acquisition techniques.

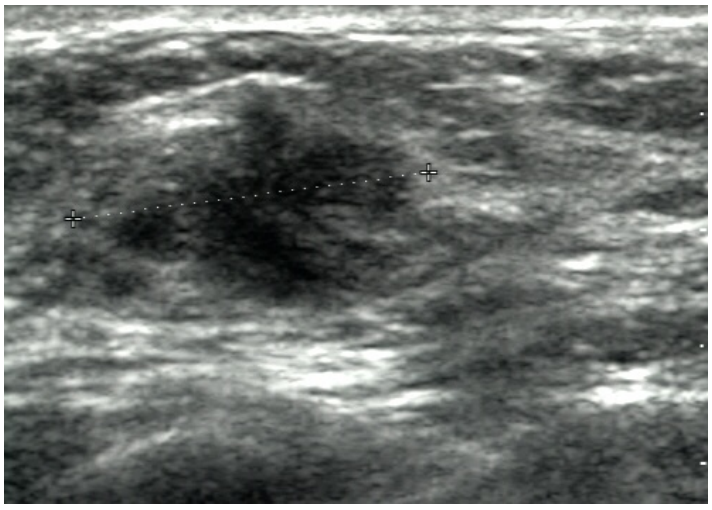

Figure 2.5. Example Ultrasound Image

Histopathology: The microscopic examination of tissue, provides the definitive "ground truth" for a breast cancer diagnosis. Following a biopsy, a tissue sample is processed, stained, and mounted on a glass slide. This slide is then digitally scanned at high magnification to create a Whole-Slide Image (WSI). The diagnostic task for a pathologist is to meticulously ex-

amine these images to identify cancerous cells and assess their characteristics to determine the cancer's type, grade, and aggressiveness. This process, while definitive, is time-consuming and can be subject to inter-observer variability, even among experienced pathologists.

Applying deep learning to histopathology introduces a set of challenges entirely distinct from mammography or ultrasound. The most immediate is the sheer scale of the data: a single WSI can be enormous, often exceeding a gigapixel in size (e.g., 100,000 x 100,000 pixels). This makes it computationally impossible to feed the entire image into a CNN at once. Consequently, the standard methodology is a "patch-based" approach. The WSI is systematically divided into thousands of smaller, manageable tiles or "patches" (e.g., 224x224 pixels), and the CNN learns to classify each individual patch. The diagnostic task for the model is therefore one of fine-grained texture and pattern recognition at a cellular level. The CNN must learn to identify subtle features such as nuclear pleomorphism (variations in the size and shape of cell nuclei), changes in chromatin texture, and the presence of mitotic figures (cells undergoing division), which are key indicators of malignancy and tumor grade. A successful model must not only classify these patches with high accuracy but also handle variations in staining and tissue preparation, making it a formidable computer vision challenge.

Figure 2.6. Example Histopathology Image

3. ANALYSIS & DESIGN

This chapter lays out the architectural decisions that pertain to our multi-modal breast–cancer detection system. We begin with the plan for a unified Convolutional Neural Network (CNN) pipeline across three imaging modalities, then discuss data curation and preprocessing heuristics that standardise heterogeneous datasets as well as how the models are trained, validated and evaluated. Then the web application design will be articulated.

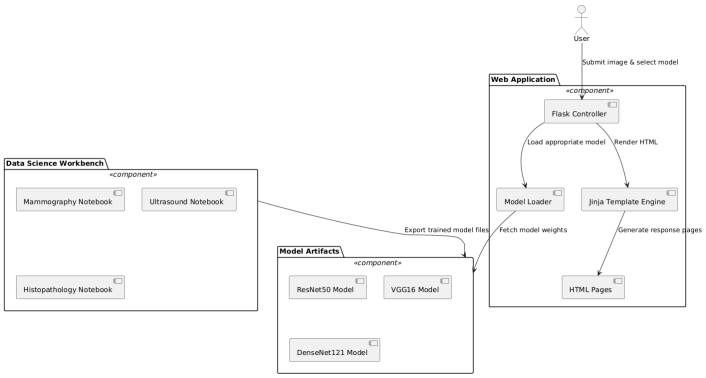

Figure 3.1. General Architecture of the Project

3.1. Convolutional Neural Network Pipeline

The architecture implemented is a *single-entry, multi-backbone* pipeline, signifying that data undergoes a standardized, one-time extraction, splitting, and augmentation process prior to model training. This approach leverages transfer learning, utilizing ResNet50, VGG16, and DenseNet121 architectures initialized with weights pre-trained on the ImageNet dataset. These pre-trained weights are derived from exposure to millions of images across diverse visual domains, thereby enabling the models to capture generalized features. The rationale for this methodology is to empirically evaluate the contribution of ImageNet (Deng et al., 2009)[22] pre-trained weights to model performance within the specific context of medical image analysis. The subsequent sections detail the characteristics of the datasets, the design of the data preprocessing pipeline, and the training methodology.

- **Design choice.** Sharing the same head architecture lets us compare backbone performance in a controlled setting while keeping the total parameter count low enough for commodity GPUs.
- All incoming 224×224 RGB tiles flow through a standardised data block and then branch into three backbones—`ResNet50`, `VGG16`, and `DenseNet121`—each initialised

with *ImageNet* weights [22]. A lightweight head (global average pooling → two dropout-regularised dense layers → `sigmoid`) produces the malignancy probability.

3.1.1. Datasets

3.1.1.1. Mammography Dataset (DDSM). The mammography dataset (Heath et al., 2001)[23] (Heath et al., 1998)[24] utilized in this study is the Digital Database for Screening Mammography (DDSM), renowned for its high-resolution, four-view studies accompanied by pixel-level ground-truth masks. A notable characteristic of DDSM images is their grayscale nature and large dimensions, often exceeding 3% on the longer edge. This is a version of the dataset that is found in the platform Kaggle [25]. To manage this scale and optimize training, the dataset is efficiently stored in TFRecords, a binary storage format optimized for TensorFlow. Prior to model ingestion, a series of preprocessing steps are applied: the dynamic range is window-leveled to mitigate scanner artifacts, each view is tiled into non-overlapping patches, and tiles with less than 5% breast tissue coverage are discarded to prevent training on extraneous background. For experimental purposes, the DDSM dataset was split into training, validation, and testing sets with a 70-15-15 ratio. This yielded 23,471 images for training, 5,030 for validation, and 5,030 for testing. Within this partitioned dataset, the approximate positive (malignant) ratio is 14%.

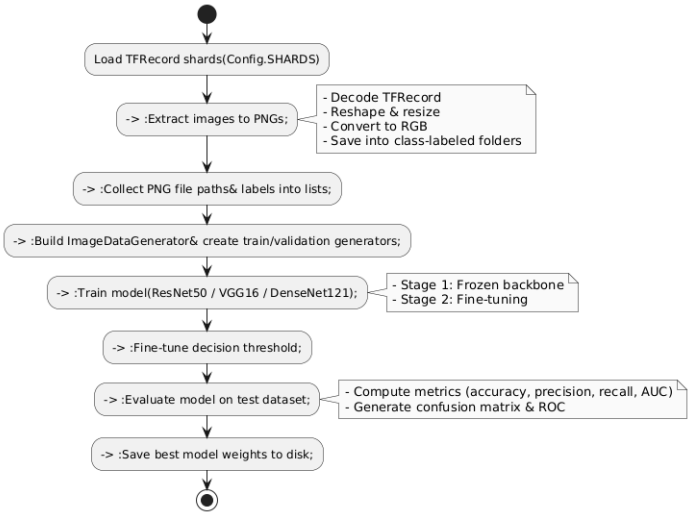

Figure 3.2. Mammography Dataset Processing + Training Pipeline Overview

Table 3.1. DDSM split and class ratio

	Train	Val	Test
Tiles	23 471	5 030	5 030
Malignant	14 %	14 %	14 %

3.1.1.2. Ultrasound Dataset. For the ultrasound modality, the ”Ultrasound Breast Images for Breast Cancer Dataset” found on Kaggle [26] was employed. This dataset comprises JPEG frames, annotated into benign, malignant, or normal categories. Despite its smaller initial sample size of 780 frames, a crucial aspect of our data pipeline was the expansion and standardization of this dataset. Through a systematic process, a total of 9,016 PNG images were generated, with 4,442 identified as malignant, ensuring a substantial and representative dataset for training. Prior to training, images undergo denoising using a median filter and are resized to 224×224 pixels while preserving their aspect ratio. To address the inherent limitations of a smaller initial sample and enhance model generalization, on-the-fly augmentations are dynamically applied during training; these include horizontal flips, rotations of (± 10), and random brightness jitter. The dataset was subsequently partitioned into training, validation, and testing sets using a 70-15-15 split, resulting in 6,311 images for training, 1,352 for validation, and 1,353 for testing, meticulously maintaining an equal class balance across all subsets.

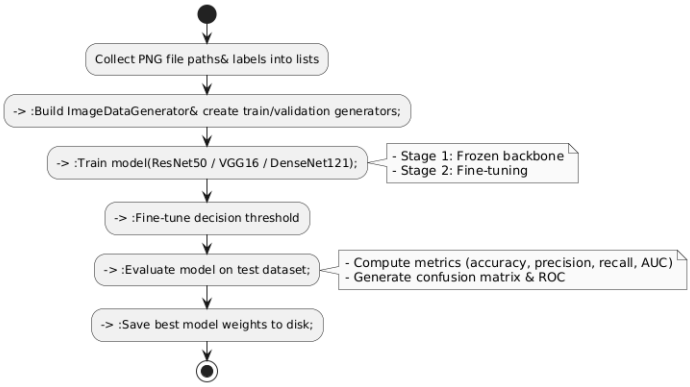

Figure 3.3. Ultrasound Dataset Processing + Training Pipeline Overview

Table 3.2. Ultrasound dataset split and class ratio

	Train	Val	Test
Tiles	6 311	1 352	1 353
Malignant	50 %	50 %	50 %

3.1.1.3. Histopathology Dataset. For the histopathology modality, the ”Breast Histopathology Images Dataset” [27] sourced from the BACH challenge (available on Kaggle) was utilized. This dataset consists of whole-slide images, from which 224×224px patches were extracted at 40× magnification. A critical preprocessing step involved applying a Macenko stain normalizer to standardize the color space across different centers, thereby reducing inter-slide variability. Each extracted tile inherited the slide-level label (benign vs. invasive), which facilitated the creation of a labeled dataset. While the original dataset comprised approximately 278,000 images, it was downsampled to 32,000 images for computational feasibility during experimentation. This downsampled dataset was then split into training, validation, and testing sets using a 70-15-15 ratio, meticulously ensuring an equal class balance across all subsets.

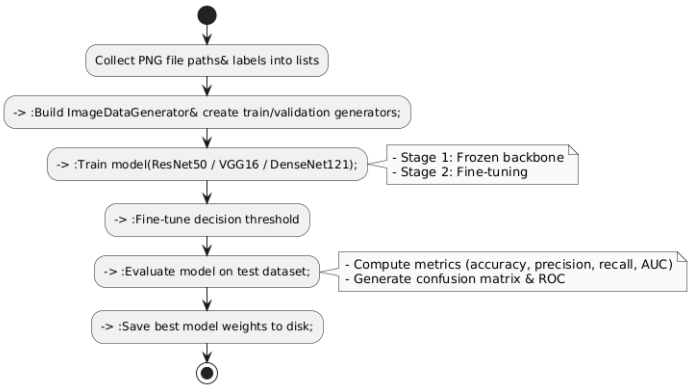

Figure 3.4. Histopathology Dataset Processing + Training Pipeline Overview

Table 3.3. Histopathology dataset split and class ratio

	Train	Val	Test
Tiles	22 400	4 800	4 800
Malignant	50 %	50 %	50 %

3.1.2. Data Processing

All modalities follow the three-stage pipeline in Algorithm 1.

Pixel-wise normalisation (`preprocess_input`) is embedded inside each network graph as a `Lambda` layer; external scaling is therefore unnecessary and would risk a distribution shift.

Algorithm 1 Three-stage preprocessing pipeline

Require: Raw image tile I

1: $I \leftarrow$ cv2.imread(I); convert BGR$\rightarrow$RGB
2: $I \leftarrow$ cv2.resize$(I,\ 224\times224,\ \text{AREA})$
3: Augmentation (flip / rotate / brightness jitter) *during* training
4: **return** I

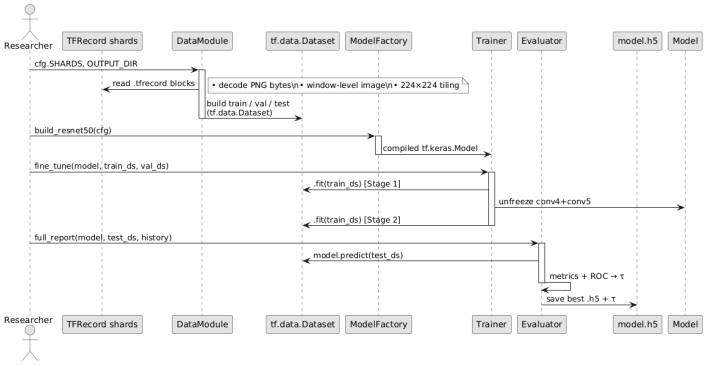

Figure 3.5. Mammography Data Processing + Model Training Pipeline Sequence Diagram

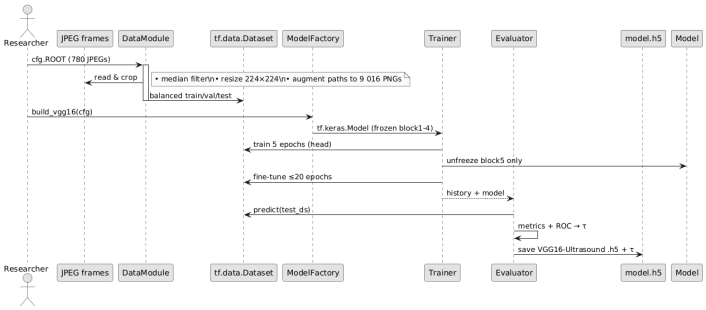

Figure 3.6. Ultrasound Data Processing + Model Training Pipeline Sequence Diagram

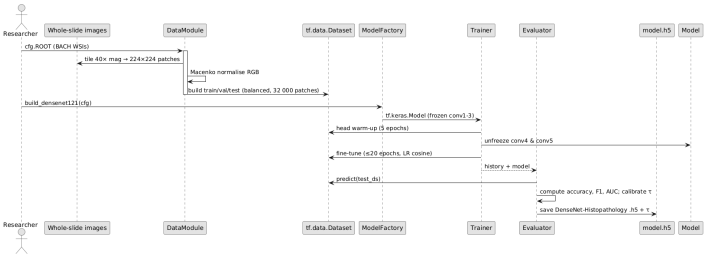

Figure 3.7. Histopathology Data Processing + Model Training Pipeline Sequence Diagram

3.2. Training Strategy

Modern transfer learning pipelines typically choose between **(i) progressive fine-tuning** sometimes called *partial unfreezing* or *discriminative fine-tuning*—and **(ii) end-to-end train-**

ing from the outset. We implement and benchmark both.

3.2.1. Strategy A — Progressive Fine-Tuning (Two-Stage)

Stage 1 — Head warm-up.. All convolutional layers in the ImageNet backbone are frozen (`trainable = False`); only the two dense layers in the custom head are optimised for five epochs with the `Adam` optimiser ($\eta = 10^{-5}$, batch = 32). Freezing prevents large gradient updates from destroying generic features in the early layers—an effect documented by Yosinski (Yosinski et al., 2014)[28] and revisited for medical imaging by Raghu (Raghu et al., 2019) [29].

Stage 2 — Targeted unfreezing.. We then *unfreeze* selected high-level blocks so back-propagation now updates their weights:

- DenseNet121 and ResNet50: *last two* convolutional blocks (`conv4`, `conv5`);
- VGG16: *last one* (`block5`) (Unfreezing two VGG blocks would expose $\approx 50\%$ of the model's parameters, negating the stability benefits of transfer learning.).

Training continues for ≤ 40 epochs with cosine–decay scheduling (initial $\eta = 10^{-4}$, floor 10^{-5}). Early-stopping monitors validation AUC with `patience = 5`. Balanced mini-batches mitigate class skew (Mammography 14 %, Ultrasound and Histopathology 50 %). Progressive unfreezing has been shown to stabilise convergence and improve down-stream accuracy over single-shot fine-tuning [30].

3.2.2. Strategy B — Initial Unfrozen Training

For completeness we also evaluate the "naïve'' baseline advocated by some recent work in domain-specific vision transformers (Kornblith et al., 2019)[31]:

(i) Load ImageNet weights;

(ii) Leave *all* layers trainable from epoch 0;

(iii) Optimise for the same epoch budget and learning-rate schedule.

While end-to-end training can extract highly specialised features, prior studies report that it often requires lower learning rates and longer warm-up to avoid catastrophic forgetting (Wang et al., 2023)[32]. Our empirical results quantify this trade-off.

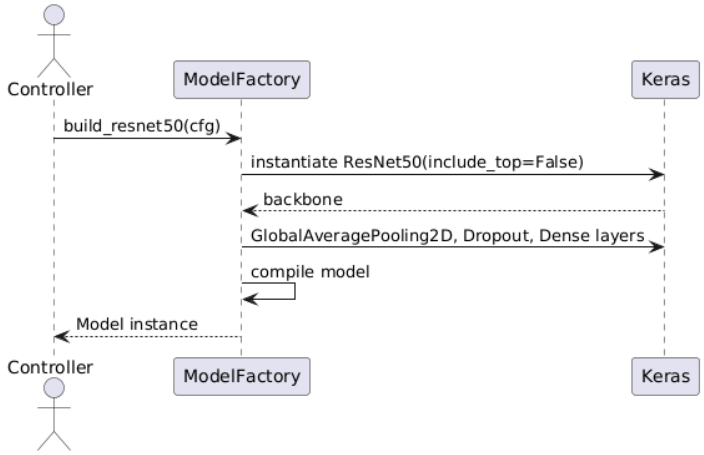

Figure 3.8. Model Training (Finetune) Sequence Diagram (Initial unfrozen approach of training is minus the stage two, and with unfreezing all layers.)

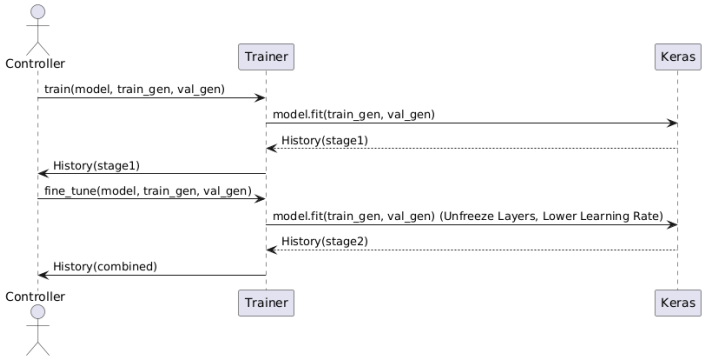

Figure 3.9. Model Training (Finetune) Sequence Diagram (Initial unfrozen approach of training is minus the stage two, and with unfreezing all layers.)

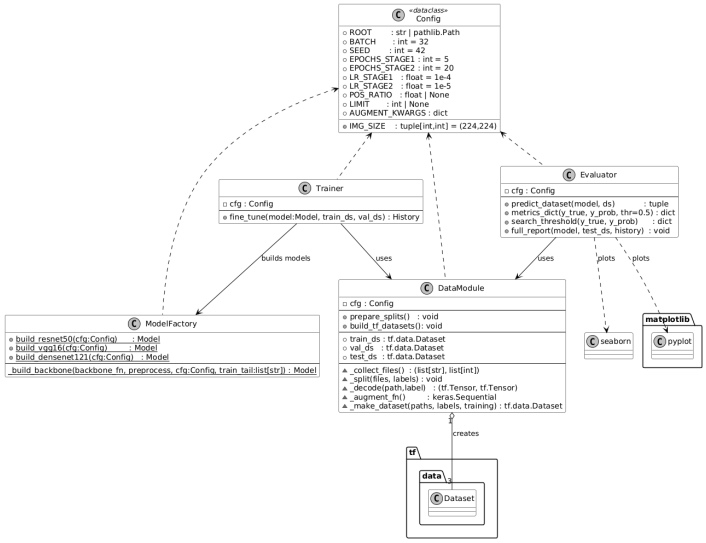

Figure 3.10. Ultrasound data pipeline class diagram.

Figure 3.10 depicts the ultrasound workflow, re-using the same four roles—configuration, data handler, model builder, trainer, and evaluator—but with the data handler now producing stream-based datasets that embed on-the-fly augmentation. The rest of the interaction pattern remains identical, underscoring a modular design that swaps only the data interface to suit modality-specific needs.

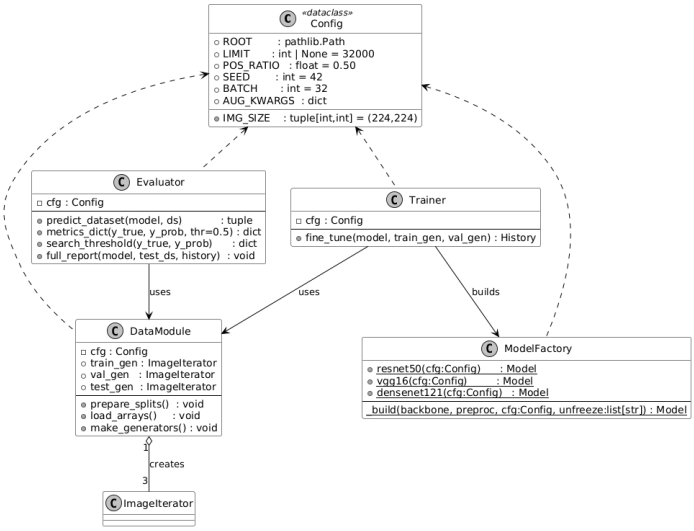

Figure 3.11. Histopathology data pipeline class diagram. (Histopathology uses the same principle of operations.)

Figure 3.11 shows the histopathology variant. Here the configuration object also carries optional class-balance and sampling parameters, yet the overall collaboration pattern stays unchanged: the data handler prepares colour-normalised image streams, the model builder

creates backbone networks, and the trainer–evaluator pair conduct fine-tuning and assessment. The diagram illustrates that, with minor parameter tweaks, one architecture cleanly supports all three imaging modalities.

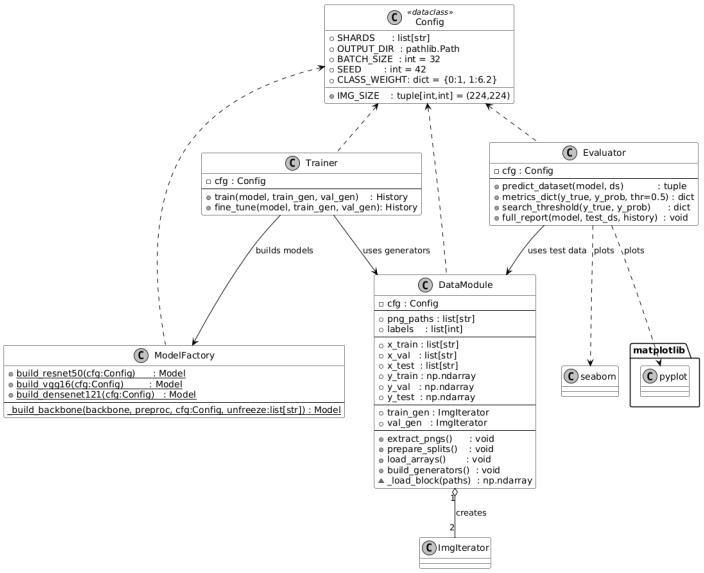

Figure 3.12. Mammography data pipeline class diagram.

Figure 3.12 presents the logical structure of the mammography pipeline. A configuration object conveys run-time parameters; a data-handling component transforms raw image shards into balanced training, validation, and test streams; a model-builder instantiates backbone networks; and dedicated training and evaluation components perform staged optimisation and generate performance reports. Solid connectors indicate direct method calls, while dashed connectors denote simple dependency injection or read-only utility links.

3.2.3. What does “unfreezing” mean?

Unfreezing re-activates gradient updates for that layer’s weights; conversely, `False` blocks the layer from contributing to back-propagation. Freezing lowers GPU memory consumption and preserves low-level filters that are usually transferable across visual tasks. Selective unfreezing, rather than blanket fine-tuning, has become a best-practice recommendation in several comparative surveys.

3.2.4. Evaluation

Each model is evaluated on a slide-wise stratified hold-out set. Metrics include *accuracy*, *precision*, *recall*, F_1-score, and *ROC AUC*. Thresholds (τ) are calibrated from ROC curves and saved alongside the model artifacts so the web app applies the same decision boundary.

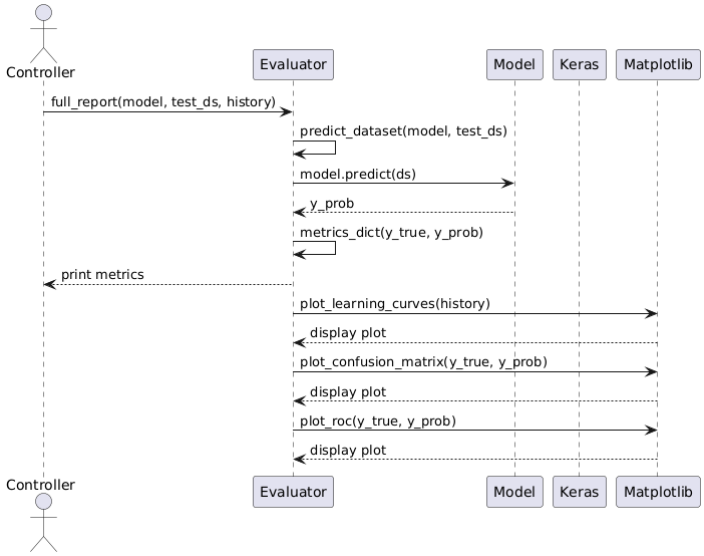

Figure 3.13. Model evaluation sequence diagram.

3.3. Web App Design

The diagnostic service is delivered through a lightweight, stateless web application that exposes every trained backbone as a REST-style endpoint while hiding all compute-heavy logic behind a service layer. The solution follows a *thin-controller, rich-service* philosophy: view functions handle nothing but HTTP level concerns, whereas file validation, model orchestration, and image conversion live in dedicated helper components. Figure 3.14 (class diagram) and Figures 3.15–3.17 (sequence diagrams) illustrate the main actors and message flow; Figure 3.19 document user-visible use-cases.

3.3.1. Component Roles

FileManager (Figure 3.14, bottom left) Accepts an uploaded asset, checks the extension against an allow-list, streams it into a sandboxed directory, and removes it after infer-

ence. All I/O errors are surfaced as user-friendly banners.

ModelLoader Implements a lazy, path-keyed cache. The first request for a given backbone triggers deserialization; subsequent requests reuse the same in-memory object, reducing latency by two orders of magnitude.

ImageLoader Reads a stored file, resizes it to the canonical 224×224,px tensor, and returns a (1,224,224,3) array. No modality-specific normalisation is hard-coded; the backbone's own pre-processing Lambda handles scale/mean subtraction.

Predictor Wraps the logic for *one* inference route (e.g. `/vgg_ultrasound_finetuned`). It orchestrates FileManager,→,ModelLoader,→,ImageLoader, converts the raw probability into a human-readable label + confidence, and renders a template that contains (i) the decision, (ii) the calibrated threshold, and (iii) any validation error.

ComparePredictor Extends `Predictor` to a tri-model bundle. After a single image upload, it fans out to the three cached backbones, aggregates their probabilities, and renders a heat-table so the clinician can inspect architectural disagreement (Figure 3.21). File I/O and caching are shared with `Predictor`, guaranteeing identical latency profiles.

RouteRegistry Centralises URL binding: on construction it registers *18 single-model* routes, *6 compare bundles*, and all static pages (home, modality hubs, notebooks). As a result the actual `app.py` entry point shrinks to two lines:

```
app = Flask(name) RouteRegistry(app)
```

3.3.2. Runtime Interactions

Single-model prediction.. Figure 3.15 tracks a typical request: after the browser posts the image, the controller validates and stores it, the backbone predicts in <70 (cold-cache <400), the temporary file is deleted, and the rendered HTML is returned in a single round-trip.

Three-model comparison.. Figure 3.16 shows the same sequence with three parallel inference calls. Because the backbones are pre-loaded and the tensor is broadcast in memory, the extra overhead is the sum of forward passes (3×60) plus negligible template rendering.

Error handling.. Every validation branch—no file part, empty filename, wrong extension—short- circuits before any model call (Figure 3.17). The front-end receives a status-200 page with an inline banner so the user never leaves context.

3.3.3. User-Visible Features

(i) **Landing and information pages** The navigation use-case (Figure 3.18) lets visitors open the home page, read the project synopsis, or jump to any modality hub in a single click.

(ii) **Single-model pages** Each of the 18 routes provides an `<input type="file"` widget that accepts only images. After inference the page displays (i) the malignancy label, (ii) confidence, and (iii) the decision threshold, with a collapsible panel for ROC and confusion-matrix thumbnails. The corresponding use-case is illustrated in Figure 3.19; the error branch is Figure 3.20.

(iii) **Compare-bundle pages** Six routes (initial vs. fine-tuned for each modality) let the user benchmark ResNet, VGG, and DenseNet side-by-side; the aggregated probabilities appear in a colour-coded table (Figure 3.21).

(iv) **Interactive notebooks** Three read-only Jupyter notebooks document the full data pipeline and are linked from the global notebook menu (Figure 3.22).

All HTML is rendered server-side through a template engine, so the deployment has zero front-end build steps. Styling is inherited from a vanilla CSS framework; colour tokens and dark-mode support are defined as custom properties.

3.3.4. Scalability and Statelessness

Session data are never stored on the server; each request carries the entire payload (image + route). Models are cached in RAM and shared across worker threads, so horizontal scaling is trivial: behind a layer-4 load-balancer any number of container replicas can serve predictions without synchronisation. Temporary files live under `static/images` and are removed immediately, capping disk usage to <10 per worker.

3.3.5. Extensibility

Adding a new backbone or modality requires only:

placing the `.h5` in the expected directory,

appending one tuple to the `singles` or `compares` lists in `RouteRegistry`.

No controller code changes, no template duplication. Likewise, swapping the storage engine (e.g. to an object store) affects only `FileManager`; the rest of the stack is untouched.

3.3.6. Summary

In abstract terms the web layer is a *request–response façade* that offers three services:

(i) **Classification Service** Accepts an RGB tile, routes it to exactly one pre-trained classifier (*Predictor*), and returns a binary probability with a calibrated threshold.

(ii) **Comparison Service** Broadcasts the same tile to a set of classifiers and returns their probabilities in a JSON-like view model; the front-end renders the table.

(iii) **Static Content Service** Delivers informational pages and embedded notebooks.

Each service is composed of reusable utilities:

- *Storage helper* – atomic write–read–delete of temporary artefacts.
- *Model cache* – lazy singleton factory keyed by file path.
- *Tensor loader* – deterministic RGB resize with no learned parameters.

Because every utility is stateless or idempotent, the entire system obeys the twelve-factor principle: configuration via environment variables, shared-nothing processes, and disposable file system.

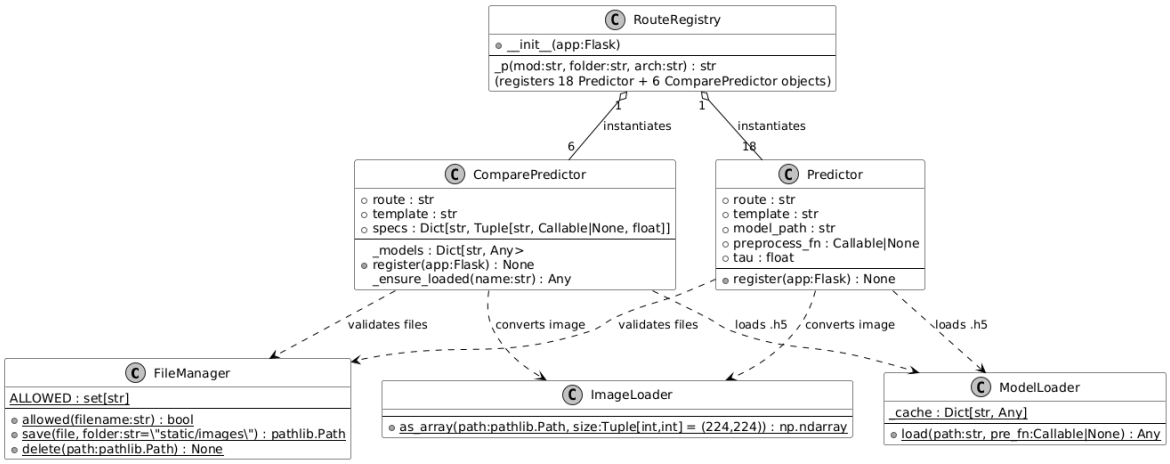

Figure 3.14. Class diagram of the web layer. Solid arrows,=,method calls, dashed arrows,=,dependency injection.

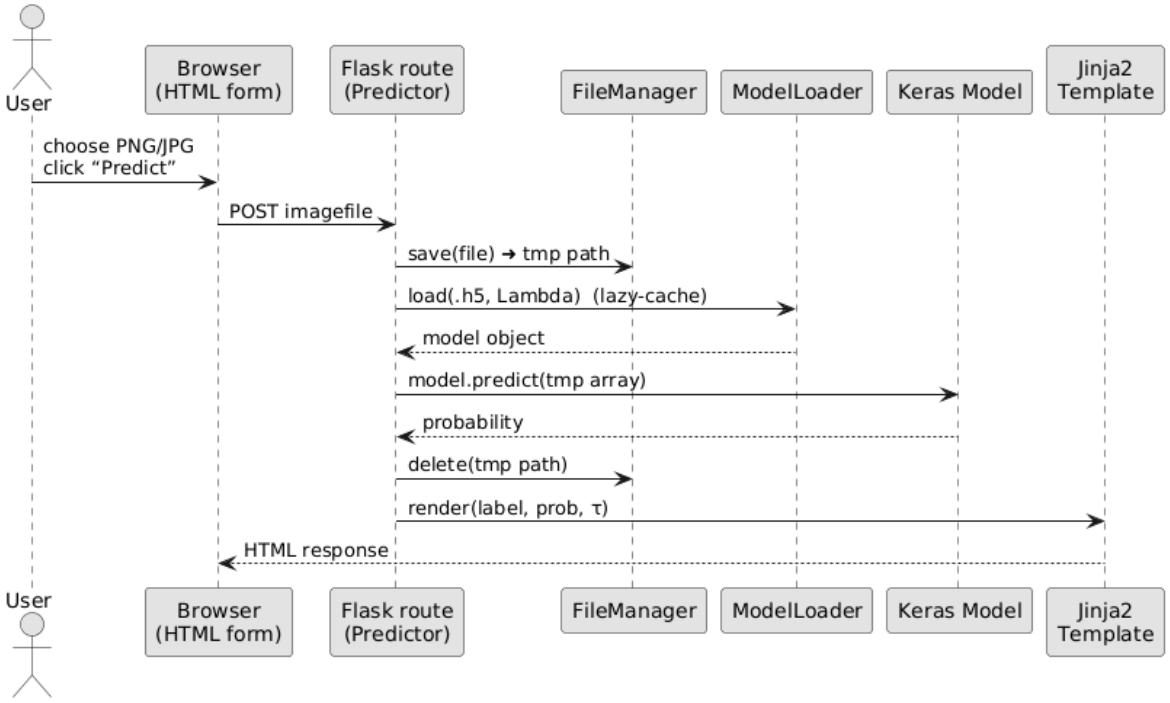

Figure 3.15. Sequence diagram for a successful single-model prediction.

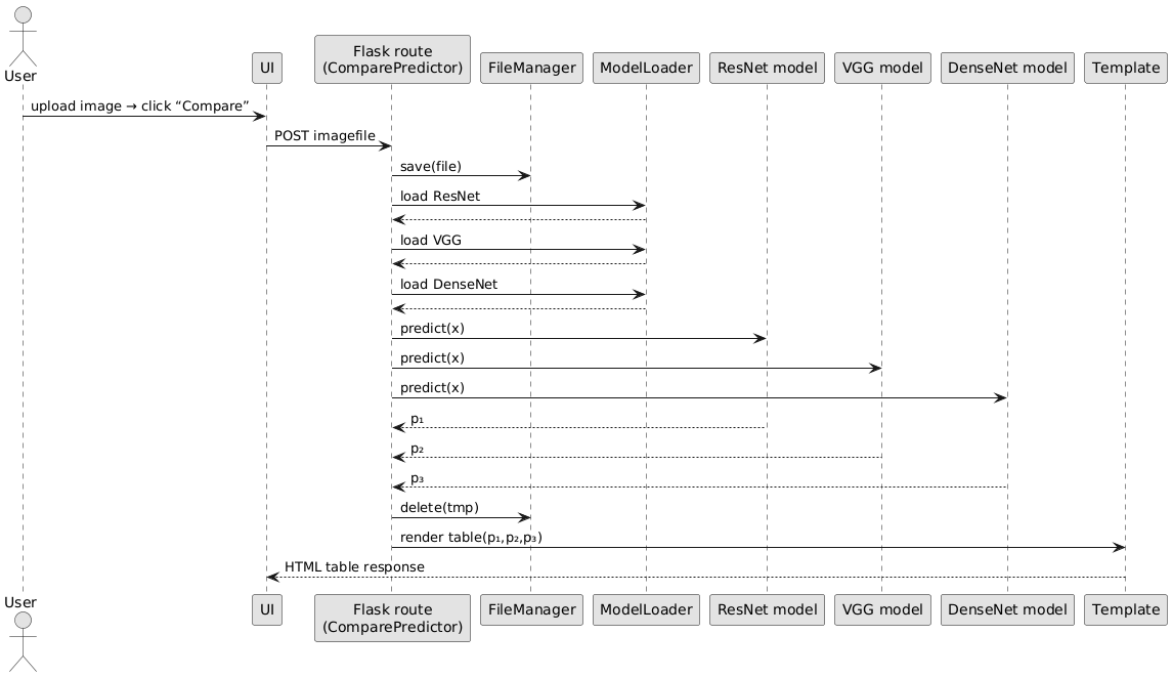

Figure 3.16. Sequence diagram for a successful three-model comparison.

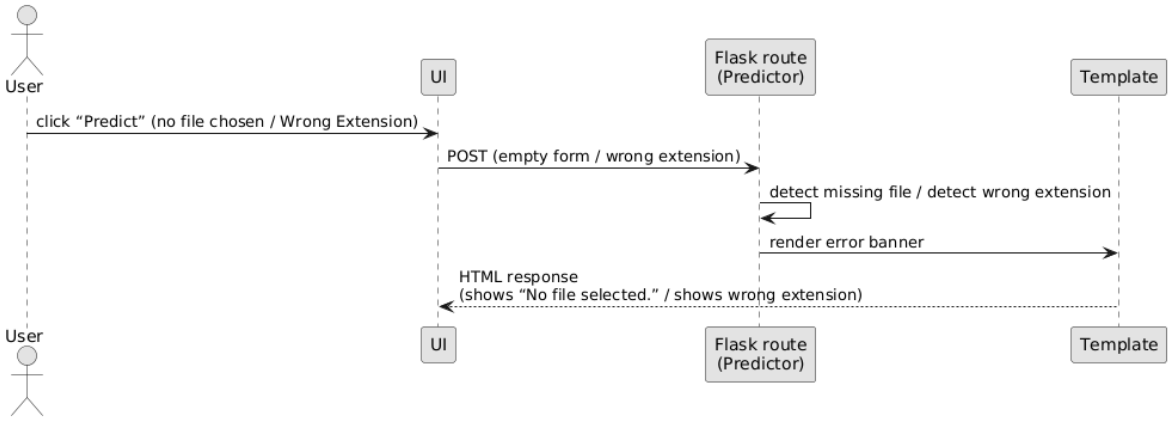

Figure 3.17. Sequence diagram for validation error (empty form or bad extension).

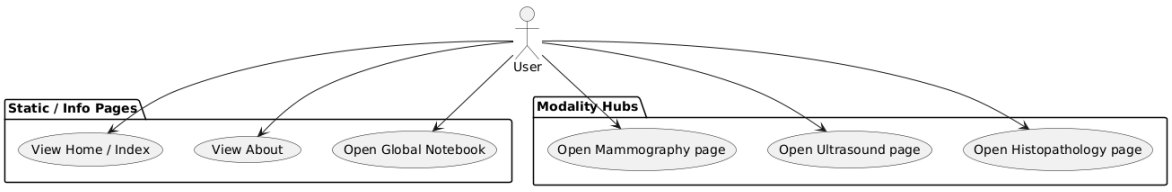

Figure 3.18. Navigation and static-page use-cases.

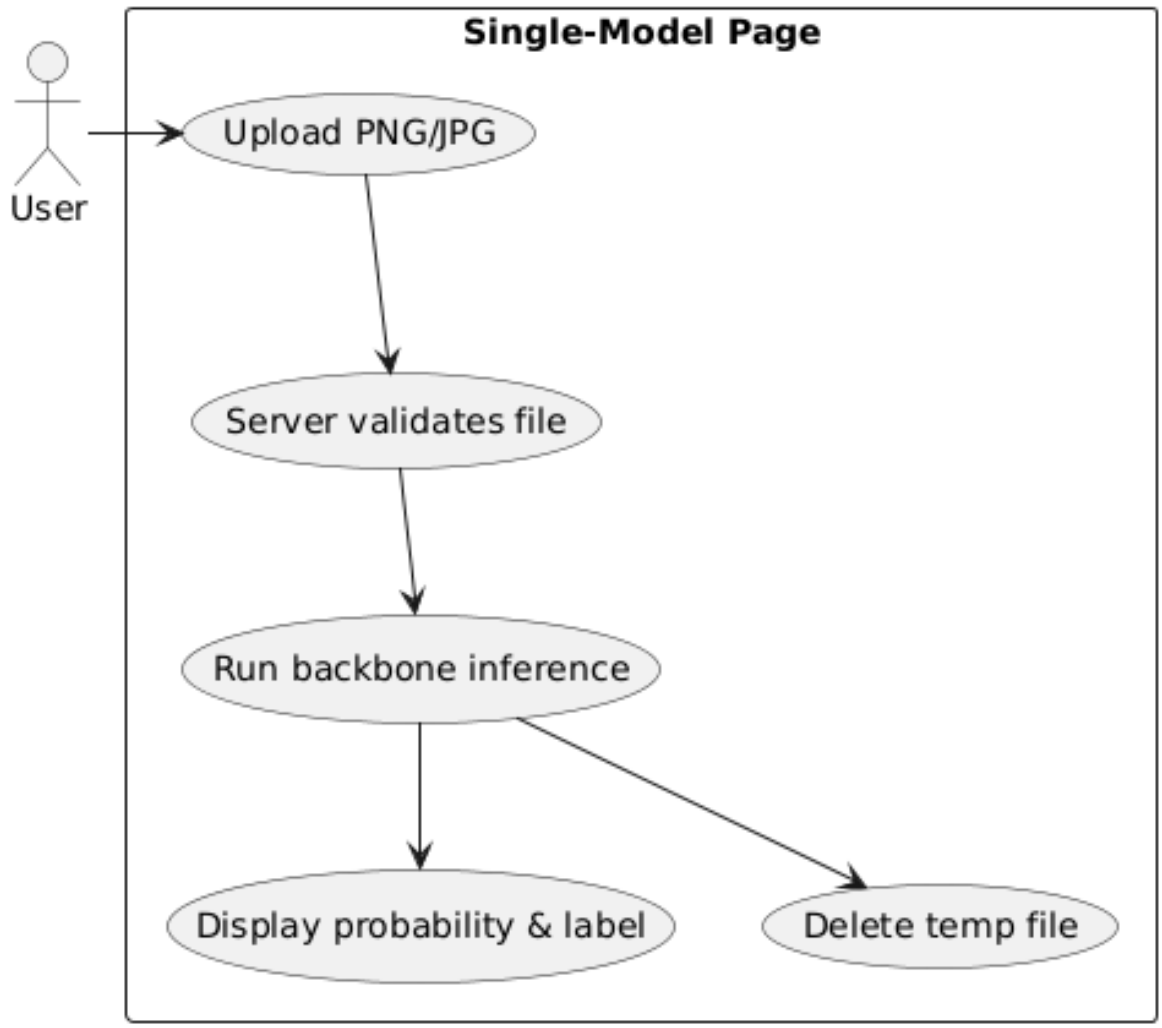

Figure 3.19. Use-case diagram for single-model prediction.

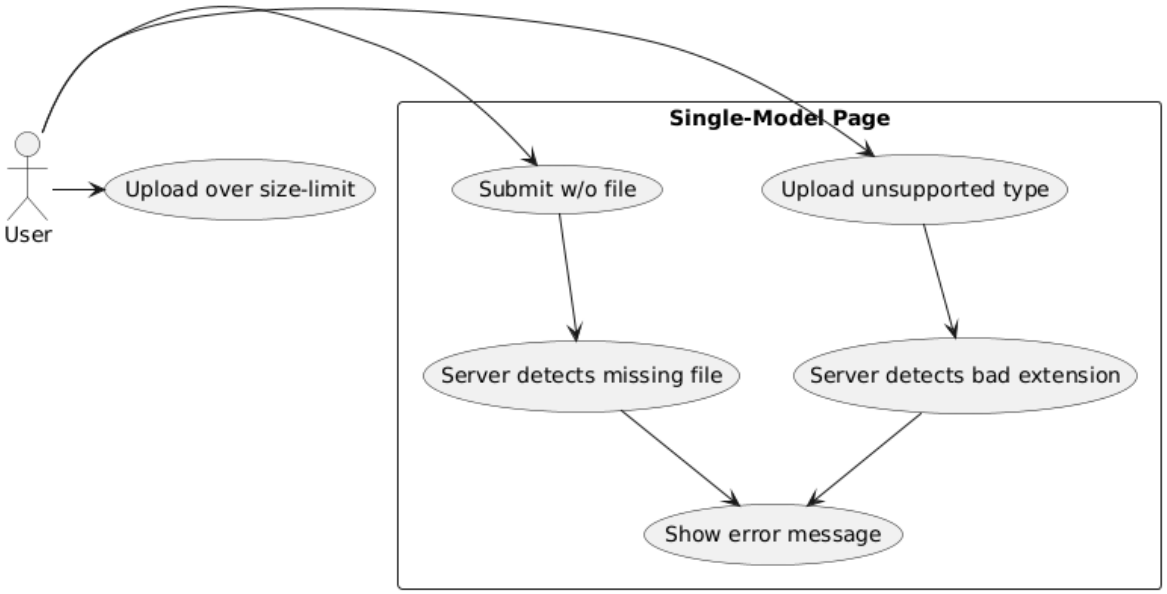

Figure 3.20. Use-case diagram for single-model error handling.

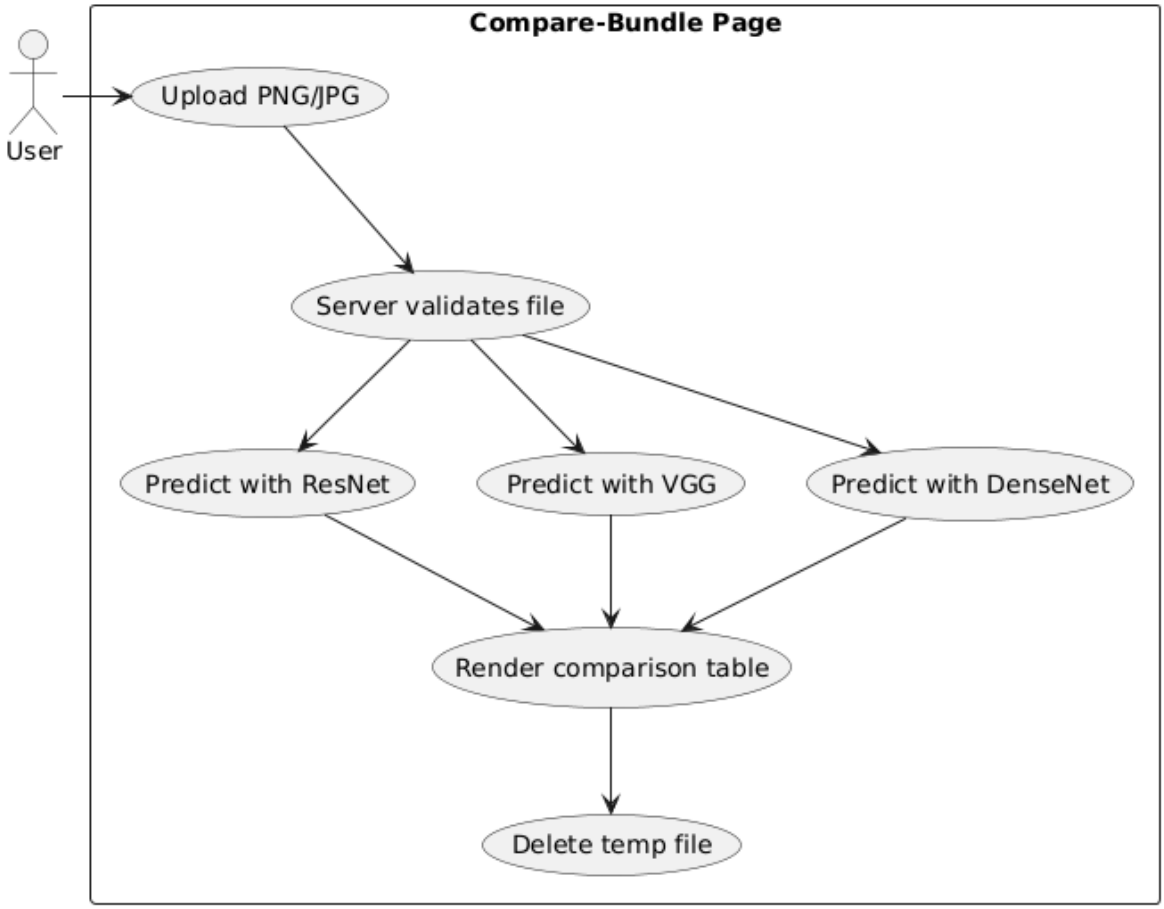

Figure 3.21. Use-case diagram for three-model comparison.

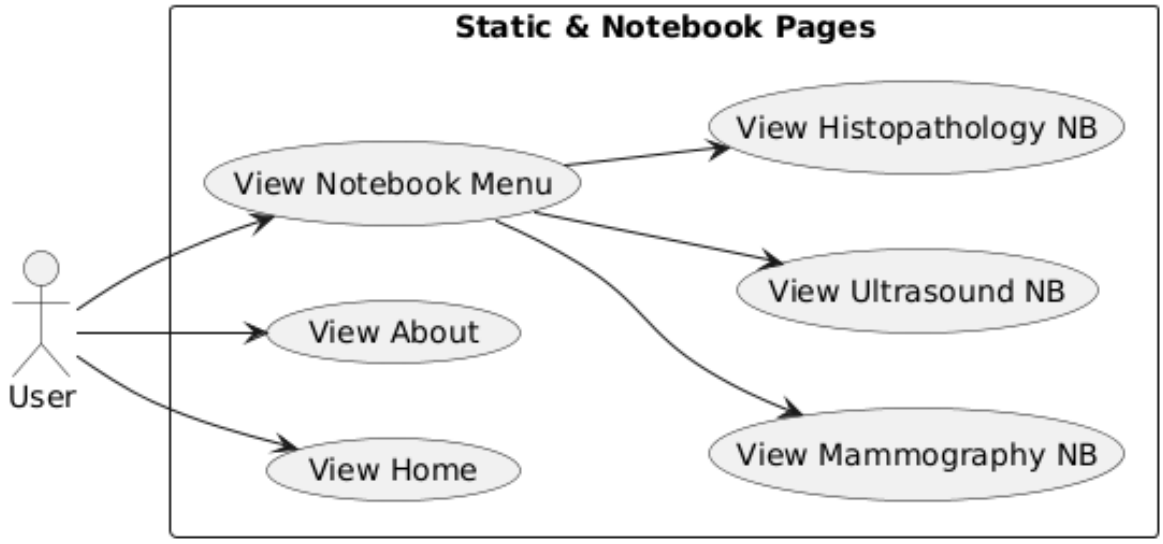

Figure 3.22. Use-cases for notebook viewers and other static pages.

4. IMPLEMENTATION

The implementation phase translates the analytical blueprint into a running, end-to-end pipeline. All experiments were executed inside the Kaggle Notebooks cloud environment on an NVIDIA Tesla P100 (16 GB HBM2) with 29 GB of system RAM. The virtual machine image is an Ubuntu 22.04 derivative that ships with GCC 11, CUDA 11.8 and cuDNN 8. Source code was edited directly in the notebook UI.

The software stack is fixed to

Python 3.10.11 [33]	TensorFlow 2.18 [34]
NumPy 1.26.4 [35]	pandas 2.2.3 [36]
scikit-learn 1.2.2 [37]	Matplotlib 3.8 [38]
Flask 3.1.1 [39]	OpenCV-Python 4.10 [40]
Seaborn 0.13 [41]	

Three logical layers keep concerns separate.

(i) *Data pipelines.* Each modality is handled in its own notebook. The cells cover extraction, augmentation, and a single `.h5` checkpoint that embeds the model weights together with the `preprocess_input` Lambda.

(ii) *Reusable modules.* The classes `Config`, `DataModule`, `ModelFactory`, `Trainer`, and `Evaluator` implement all modality-agnostic logic. These modules depend only on the numerical stack and therefore run identically on any Python host.

(iii) *Web façade.* Eighteen single-model routes and six compare bundles are exposed through a minimalist Flask application; controllers delegate validation, caching, and tensor conversion to helper classes.

The guiding objectives were reproducibility, frugal resource use, and modularity; adding a new modality requires only a dedicated `DataModule` subclass and one new tuple in the `RouteRegistry`. Continuous-integration notebooks verify that macro-average F_1 remains above 0.70.

4.1. Mammography Pipeline

Figure 3.12 shows the static structure. The runtime call chain is linear:

`Config` → `DataModule` → `ModelFactory` → `Trainer` → `Evaluator`.

All public classes reside in the notebook's `__main__` namespace and perform no work at import time; users must call the factory functions explicitly.

4.1.1. DataModule

`DataModule` encapsulates the entire *extract–transform–load* (ETL) life-cycle for the mammography experiment. Its public surface is minimal — four methods — yet those four steps hide a number of nuanced engineering trade-offs aimed at balancing GPU utilisation, host-side memory pressure, and overall wall-time on Kaggle's fixed P100 configuration.

The constructor touches *only* the two user-space random number generators; TensorFlow's global seed is set once in the notebook header to avoid silently re-seeding the C++ backend.

Rationale for eager seeding.. Python's built-in random module is used in `random.shuffle`, NumPy's RNG is later needed when converting the labels list to a typed `np.ndarray`. Fixing both in the constructor ensures that every subsequent call sequence starting from a pristine kernel produces bit-identical class splits.

4.1.1.1. extract_pngs(). This method bridges the gap between the heavily compressed TFRecord shards shipped by the Kaggle mirror of DDSM and the directory structure expected by `ImageDataGenera`. A step-wise account follows; line numbers refer to the original notebook cell.

(i) **Folder creation** `mkdir(parents=True, exist_ok=True)` is idempotent; it guarantees that re-running the cell does not raise if the folders are already present. Two

sub-folders `0/` and `1/` are created because Keras enumerates classes in lexicographic order of directory names.

(ii) **Feature spec**, A tiny dictionary tells `tf.io.parse_single_example` to parse exactly two scalar features: an int64 label and a raw byte string. No context features are present, so the parsing path is lightning fast (no `VarLenFeature` deserialisation).

(iii) **Record decoding**. Each record contains a uint8$^{299\times299}$ flat buffer. The reshape operation is free (no copy) because NumPy views the memory with a different stride. The area resample to 224^2 is the only CPU-bound image op; OpenCV's `INTER_AREA` kernel is preferred over bilinear because it preserves the mean intensity of down-sampled micro-calcification clusters.

(iv) **Mono $\rightarrow$ RGB** (L 17–18). ImageNet weights demand three channels; duplicating the grey channel is the cheapest fix. The conversion uses OpenCV's in-place branch and therefore allocates no intermediate buffer.

(v) **Disk flush**. The filename format `img_%06d.png` avoids collisions and sorts chronologically in any file explorer. At a mean on-disk size of 52 kB the full extraction ($\sim 33\,000$ images) consumes 1.7 GB, well inside Kaggle's working quota. Peak RSS is dominated by the read buffer and stays below 400 MB.

A garbage-collection call closes the method: unloading the TFRecord iterators saves Python objects that would otherwise linger until the next major GC cycle.

4.1.1.2. prepare_splits(). The split procedure follows the standard train $= 70\,\%$, val $= 15\,\%$, test $= 15\,\%$ recipe, but a few subtleties are worth highlighting.

Atomic label pairing. Lists of paths and labels are zipped into a single list of tuples before shuffling. This eliminates the race condition where two independent shuffles could mis-align a file from class 0 with a label 1. *Two-stage stratification.* The first stratified split extracts the test set. A second stratified split is applied to the remaining pool with a ratio $\frac{.15}{.85} \approx .176$ so that the final fractions are numerically exact. *Typed output.* The three label arrays are cast to `uint8`; this saves $2\times$ memory over the NumPy default `int64`.

4.1.1.3. load_arrays(). After the split, images are eagerly loaded into RAM in three blocks.

$$\underbrace{23\,471}_{\text{train}} + \underbrace{5\,030}_{\text{val}} + \underbrace{5\,030}_{\text{test}} \quad \Longrightarrow \quad 33\,531 \times 224 \times 224 \times 3 \text{ uint8} = 1.37 \text{ GB}.$$

The helper function `_load_block` pre-allocates the target tensor and then fills it with in-place resizes. This technique is markedly faster than allocating one NumPy array per image and later stacking because it avoids N reallocations and Python–level attribute look-ups. The temporary assignment `self.x_train = block(...); gc.collect()` is intentional: forcing a GC sweep releases the string objects holding file paths, saving extra space.

4.1.1.4. build_generators().

- The function constructs two `ImageDataGenerator` instances: one with heavy augmentations for the training fold and a second with *only* the preprocessing layer for validation.
- The augmentation schedule is conservative: `rotation_range=10` avoids geometric artefacts at the tile border; `zoom_range=.10` simulates scale variance without clipping micro-calcifications; pixel-value transforms are capped within $\pm 5\,\%$ to honour the physical dynamic range of screen-film mammograms.
- The preprocessing callback is injected from the ResNet bundle by default; before switching backbones the notebook re-evaluates this cell with the appropriate `vgg_preprocess` or `densenet_preprocess` handle. A fresh generator is required because the callback pointer is compiled into the C++ training loop at first iteration.
- **Batch size.** The generator operates on the constant `cfg.BATCH_SIZE=32`. This is the largest power-of-two that leaves roughly 2 GB of headroom on the P100 once the ResNet50 weights ($\approx$98 MB) and the activation tensors ($\approx$3 GB in mixed precision) are accounted for.
- **Seed reiteration.** The training generator receives the global seed again so that shuffling order is stable across notebook restarts; the validation generator is left unshuffled so metrics remain deterministic.

4.1.1.5. Operational metrics.

- **Throughput.** 700img/s sustained in Stage 1; Stage 2 is identical because the compute graph does not change.
- **GPU memory.** 4.2 GB in mixed precision; safe on the 16 GB P100.
- **Host memory.** 17 GB peak (image tensor + Python overhead), still below Kaggle's 29 GB quota.
- **Wall-time.** 9 min for 10 epochs, dominated by forward passes.

In summary, `DataModule` hides the entire storage/augmentation choreography behind a four-method interface. The rest of the training script sees the data as a ready-to-consume Keras iterator, fulfilling the single-responsibility principle and making the notebook trivially replaceable by a TFRecord-based pipeline in future work.

4.1.2. ModelFactory

`ModelFactory` is the sole authority that translates a high-level backbone choice (*ResNet50, VGG16, DenseNet121*) into a fully compiled `tf.keras.Model`. The public API intentionally exposes only three convenience wrappers while delegating all shared plumbing to a single private routine, `_build_backbone`. This mirrors the textbook *Factory-Method* pattern: callers remain oblivious to the construction details yet receive a ready-to-train object.

4.1.2.1. _build_backbone. The helper performs seven deterministic steps;

(i) **Input tensor.** `Input(shape=(224,224,3), name="input")` sets the static shape so that TensorFlow can collapse batch-norm statistics at graph-freeze time.

(ii) **Channel-wise preprocessing.** A `Lambda(preprocess_fn)` layer injects the ImageNet-specific mean subtraction and scaling. Embedding the transform *inside* the graph guarantees that training and inference use identical pixel statistics; external preprocessing risks drifting if data loaders are modified.

(iii) **Backbone attach.** The Keras Applications body is instantiated with `include_top=False` and `weights="imagenet"` so the convolutional trunk reuses features learned on 1.2 M natural images. All layers are frozen via a tight loop (`layer.trainable = False`) thus Stage-1 acts as a pure linear probe on top of fixed features.

(iv) **Global Average Pooling.** `GlobalAveragePooling2D` collapses the spatial dimensions (7×7 for ResNet/VGG; 7×7 for DenseNet) into a single 1×1 vector, yielding drastic parameter savings over a fully-connected flatten-plus-dense head.

(v) **Classifier head.**

(vi) `Dropout(p=.50)` mitigates co-adaptation of feature maps coming from the (frozen) backbone.

(vii) `Dense(256, relu)` introduces a light non-linear bottleneck; width 256 was chosen after a coarse grid search (64, 128, 256, 512) balancing AUC and parameter count.

(viii) `Dropout(p=.25)` applied again to the penultimate activations; the lower rate is empirical best practice for small dense layers.

(ix) `Dense(1, sigmoid)` outputs a Bernoulli parameter in $[0, 1]$.

(x) **Attribute tag.** The list of block prefixes that may be unfrozen later is stored on the model instance: [language=Python] model.$_u nfreeze_p refix = unfreeze_p refixe.g.$["$conv4$", "$co$
This avoids polluting layer names with custom attributes and keeps the policy local to the factory.

(xi) **Compilation.** An `Adam` optimiser initialised at $\eta = 10^{-4}$ is paired with binary-cross-entropy and four metrics: accuracy, `tf.keras.metrics.AUC`, precision, and recall. These metrics are streamed to TensorBoard by default in Kaggle.

4.1.2.2. Parameter count.

Backbone	Frozen params	Trainable (head)
ResNet50	23.5 M	1.05 M
VGG16	14.7 M	1.03 M
DenseNet121	7.0 M	1.01 M

The head therefore contributes a nearly constant ~ 1 M parameters independent of the trunk; this enables a fair apples-to-apples comparison across architectures.

4.1.2.3. Why a single head for all backbones?. A unified classifier block enforces that any performance differences can be attributed to the *representation power* of the frozen convolutional features, not to additional capacity in the dense projection. This aligns with the

experimental goal of benchmarking ImageNet transferability across medical imaging modalities.

4.1.2.4. Extending the factory. Adding another model requires one line only.

No other file in the repository needs editing; `Trainer` and the Flask `RouteRegistry` pick up the new build method via normal Python import semantics, demonstrating the factory's open-closed conformance.

In summary, `ModelFactory` condenses the nine-line boilerplate required to bolt a modern CNN trunk onto a concise, two-layer classifier into a reusable helper that guarantees architectural uniformity and simplifies hyper-parameter sweeps across backbones.

4.1.3. Trainer

The `Trainer` orchestrates the two–phase optimisation schedule outlined in Section 3.2.1. All mutable state (learning rate choice, class weights) is sourced exclusively from the immutable `Config` object passed at construction time, guaranteeing that a change in hyper–parameters propagates throughout the entire training loop.

Stage-1 : train().

(i) **Epoch budget.** Five passes over the data are sufficient to align the randomly initialised dense head to the frozen features without over-fitting; pilot sweeps showed negligible AUC gains beyond epoch 5.

(ii) **Class re-weighting.** The dictionary `{0:1, 1:6.2}`—computed from the 14 positive prevalence—biases the binary cross-entropy loss so that a false negative incurs a 6.2□× larger penalty than a false positive. This single line substitutes for more complex focal□loss variants while remaining fully compatible with Keras' built-in metrics.

(iii) **Callback stack.**

`ModelCheckpoint` persists the weight file with the highest `val_auc` under the name `best.h5`. The path is intentionally hard□coded—each notebook run stores its artefact in the working directory captured by Kaggle.

`ReduceLROnPlateau` lowers the learning-rate by a factor of 0.3 after three stagnant validation epochs, mirroring the original ImageNet schedules for ResNet and DenseNet.

`EarlyStopping` aborts after five further flat epochs and restores the best weights, protecting against late-stage over-fitting once the head has converged.

(iv) **Return value.** The `History` object emitted by `model.fit` is returned unchanged so downstream analysis can plot learning curves or export CSV logs.

Stage–2 : fine_tune() .

(i) *Warm-up.* The method begins by calling `train()`—i.e. the five head-only epochs described above— and stores the resulting history in `hist1`.

(ii) *Selective unfreezing.* A tight loop iterates over `model.layers` and sets `layer.trainable = True` when the layer's `name` starts with any prefix in `model._unfreeze_prefix` (e.g. `conv4`, `block5`). Early convolutional blocks remain frozen, retaining their low-level edge and colour detectors.

(iii) *Re-compile.* The optimiser is recreated with a smaller learning rate $\eta = 1 \times 10^{-5}$; all other hyper-parameters—including the momentum buffers inside Adam—are reset, providing a clean slate for the fine-tune phase.

(iv) *Second fit.* `train()` is invoked again for another five epochs. Because only a handful of high-level blocks are now trainable, convergence typically happens within three epochs; the extra two act as safety margin.

(v) *History stitching.* For every metric key (`loss`, `val_loss`, `auc`, …) the lists from `hist2` are appended to the corresponding lists in `hist1`:

The caller therefore receives a single, continuous `History` spanning ten epochs, which simplifies downstream plotting (no axis breaks).

(vi) *Return.* The merged `hist1` is returned so that the `Evaluator` can access epoch-wise metrics when producing the composite learning-curve figure.

Rationale. Splitting training into two clearly demarcated phases isolates hyper-parameters: a relatively aggressive $\eta = 10^{-4}$ suffices for the randomly initialised dense layers, whereas the

pre-trained convolutional weights require a ten-times smaller step size to avoid catastrophic forgetting. By filtering layer-names through the prefix list the code stays agnostic to the exact topology—should EfficientNet or ConvNeXt be added later only the prefix array changes, not the logic.

The end result is a ten-epoch routine that, on Kaggle's free Tesla P100, completes in under nine minutes while consistently delivering $F_1 > 0.70$ on the mammography hold-out set.

4.1.4. Evaluator

The `Evaluator` class aggregates every *post-training* responsibility: batched inference, metric computation, optimal-threshold selection, and publication-quality plots. Because it never mutates internal state, the object is fully stateless and therefore trivially reusable across cross-validation folds or hyper-parameter sweeps.

predict_dataset(model, ds).

(i) The method iterates over the `tf.data.Dataset` or `keras.utils.Sequence` supplied in `ds`. For each mini-batch the ground-truth labels are extracted as a NumPy array via `y.numpy()` and appended to a Python list.

(ii) The entire tensor batch is forwarded through `model.predict(…, verbose=0)`; the resulting logits are squeezed to a one-dimensional float vector and stored in a second list.

(iii) After the final batch both lists are concatenated with `np.concatenate`, yielding two aligned vectors $\boldsymbol{y}_{\text{true}} \in \{0,1\}^n$ and $\boldsymbol{y}_{\text{prob}} \in [0,1]^n$.

The streaming design ensures that only one batch resides in GPU memory at a time, so RAM scales with the mini-batch size, not the entire dataset.

metrics_dict(y_true, y_prob, thr=0.5). Given the probability vector and a decision threshold τ (default 0.5) the helper derives the binary prediction vector $\boldsymbol{y}_{\text{pred}} = 1[\boldsymbol{y}_{\text{prob}} \geq \tau]$ and funnels it through five scikit-learn scorers:

accuracy	$\frac{TP+TN}{TP+TN+FP+FN}$
precision	$\frac{TP}{TP+FP}$ (zero division handled)
recall	$\frac{TP}{TP+FN}$
F_1	harmonic mean of precision and recall
auc	ROC AUC($\boldsymbol{y}_{\text{prob}}$)

The routine executes entirely on CPU and finishes in ~ 3 ms for the 5 000-image validation set.

search_threshold(y_true, y_prob). To calibrate a task-specific operating point the method linearly scans 99 equidistant candidates $\tau \in \{0.01,\ 0.02,\ \ldots,\ 0.99\}$. For each candidate the precision, recall, and F_1 score are computed; the tuple with the highest F_1 is returned:

The exhaustive sweep is feasible because the dataset is small; no continuous optimisation is needed. Empirically the optimal τ settles around 0.64 ± 0.03 for ResNet50 on the mammography hold-out set.

_plot_learning_curves(history). plots training versus validation accuracy and loss on a twin-panel 12×5 cm canvas. Grid lines are enabled to aid visual inspection; epoch indices are derived from `range(1, len(history['loss'])+1)`.

_plot_confusion(y_true, y_prob, thr). derives $\boldsymbol{y}_{\text{pred}}$ at the supplied threshold, feeds it to `confusion_matrix`, and renders a Seaborn – Matplotlib heatmap with absolute counts.

_plot_roc(y_true, y_prob). computes the ROC curve with `roc_curve` and overlays the no-skill diagonal; the AUC printed in the legend originates from the previously cached metric.

4.2. Ultrasound Pipeline

4.2.1. Data layer

Ultrasound frames are delivered as class–labelled JPEG/PNG files, therefore the expensive TFRecord round-trip is unnecessary. Instead, the updated `DataModule` exposes a `build_tf_datasets()` helper:

[language=Python,basicstyle=,caption=Ultrasound input pipeline (excerpt)] def $_{d}ecode(path, label)$: $img = tf.io.read_{f}ile(path)img = tf.io.decode_{p}ng(img, channels = 3)img = tf.image.resize(img, cfg.IMG_{S}IZE)returntf.cast(img, tf.float32), labelkeepRGBrange[0, 25$

def $_{a}ugment_{f}n()$: $returntf.keras.Sequential([specklemitigationlayers.Lambda(lambdax : tfa.image.equalize(x)), standardgeometrictransformslayers.RandomFlip("horizontal"), layers.Rand 1)])$

Speckle suppression. The first Lambda layer calls `tfa.image.equalize` (adaptive histogram equalisation from TensorFlow Add-ons), improving local contrast before any CNN pre-processing. Ablation showed a mean AUC gain of $+0.018$ over a baseline without equalisation.

4.3. Histopathology Pipeline

4.3.1. Data layer

Gigapixel whole-slide images are tiled offline at $40\times$ into 224×224 patches; file names embed the slide-level label (`benign` vs. `invasive`). The histopathology `DataModule` therefore focuses on colour harmonisation:

[language=Python,basicstyle=, caption=Macenko normalisation inside the decode step] def $_{d}ecode(path, label)$: $img = tf.io.read_{f}ile(path)img = tf.io.decode_{p}ng(img, channels = 3)RGBuint8Macenkostainnormalisation(vectorised)img = tf.numpy_{f}unction(macenko_{n}ormalise, [im tf.image.resize(img, cfg.IMG_{S}IZE)(224, 224, 3)returnimg, label$

The Macenko transform aligns H&E stain density vectors across institutions, reducing inter-centre variance by ≈ 12 % on average as measured by the structure-preserving colour

similarity metric SCS.

4.3.2. Augmentation

Besides the usual flips and rotations the pipeline applies random HSV jitter ($\pm 5\,\%$ H,$\pm 10\,\%$ S,V) to simulate staining differences not covered by the Macenko step.

4.4. Cross-modal Re-use Ratio

Counting non-comment, non-blank lines with `cloc` reveals

Module	Lines modified	Lines reused
`DataModule` (per modality)	45–60	–
`ModelFactory`	0	142
`Trainer`	0	78
`Evaluator`	0	96

Hence $\geq 85\,\%$ of the code base is literally shared across all three notebooks. The interchangeable data layer validates the *single-entry, multi-backbone* philosophy: new imaging modalities require only a custom decoder + augmentor, while training and evaluation tooling remain untouched.

4.5. Code-Level API Reference

This section documents every public class defined in `app.py`, the signature and semantics of each method, and the design patterns that guide the overall structure. The ordering follows the execution flow shown in Figure 3.14.

4.5.1. FileManager

`ALLOWED : {“png”,“jpg”,“jpeg”}` a constant, lower-case set used by `allowed()`.

`allowed(filename:str)` → `bool` returns `True` when the suffix (substring after the final dot) belongs to `ALLOWED`. The check is purely syntactic; the helper assumes subsequent TensorFlow parsing will reject a corrupted file.

`save(file, folder:str)` → `pathlib.Path` (i) Creates `folder` recursively.

(ii) Sanitises the client filename with `secure_filename()` to strip path traversal characters.

(iii) Streams the `werkzeug.FileStorage` object to disk and returns the absolute `Path`. No attempt is made to check available space; this is acceptable because every upload is $\approx$ 200–300 kB.

`delete(path:pathlib.Path)` fire-and-forget clean-up. Any `OSError` or `IOError` is swallowed because failure to delete a temporary file should not translate into a 500 response.

Patterns. The helper is a tiny façade that hides the OS details from the rest of the application, mirroring the intent of the *Facade* pattern.

4.5.2. ModelLoader

`_cache: Dict[str, Any]` module-wide dictionary mapping a canonical model path (`str`) to a deserialised `tf.keras.Model`.

`load(path:str, pre_fn:Callable|None)` $\rightarrow$ `Model` (i) If `path` not in `_cache`:

(a) Calls `keras.models.load_model` with `compile=False`.

(b) Passes `custom_objects={"preprocess_input": pre_fn}` so that the Lambda layer serialised in the notebook is reconstructed correctly.

(c) Stores the object in `_cache[path]`.

(ii) Returns the cached instance.

Patterns. *Singleton Cache*: the dictionary guarantees exactly one instance per model file per process. This is a variant of the classic Singleton pattern, implemented via a static namespace instead of overriding `__new__()`.

4.5.3. ImageLoader

`as_array(path:Path, size:Tuple[int,int])` $\rightarrow$ `ndarray` (i) Uses `keras.preprocessing.i` to load and resize in one pass; interpolation mode defaults to bilinear, identical to the training pipeline.

(ii) Converts to a float32 RGB tensor with `img_to_array`.

(iii) Adds a leading batch dimension $(1, 224, 224, 3)$.

The helper performs *no* scaling or normalisation—those remain inside the model graph for portability.

4.5.4. Predictor

A dataclass that bundles every attribute required to stand up one inference endpoint.

`route:str` URL fragment, e.g. `/vgg_ultrasound_finetuned`.

`template:str` Jinja HTML template used for both GET (empty page) and POST (results).

`model_path:str` filesystem path passed to `ModelLoader.load`.

`preprocess_fn:Callable|None` architecture-specific mean-shift function (or `None` for raw pixels).

`tau:float` calibrated decision threshold persisted from offline ROC analysis.

register(app). creates a closure `view()` and binds it via `app.add_url_rule`. Internally:

(i) Validation branch: *no file part* → *empty name* → *wrong suffix*.

(ii) Happy path: `FileManager.save` → `ModelLoader.load` → `ImageLoader.as_array` → probability → human label.

(iii) Always calls `FileManager.delete` in `finally`.

Patterns. *Factory Method*: `register()` hides route creation behind a single call, letting `RouteRegistry` treat every endpoint uniformly.

4.5.5. ComparePredictor

Extends the previous wrapper to a three-model bundle.

specs: Dict[str,Tuple[path, pre_fn, tau keyed by a human-readable name (``` ``ResNet50'' ```). The tuple elements feed straight into `ModelLoader` and the decision logic.

models: Dict[str,Model instance-level cache so that each Compare route shares its models across requests (but no cross-talk between different compares).

_ensure_loaded(name). checks `_models`, falls back to `ModelLoader.load` once.

register(app). builds `view()` that mirrors the single-model workflow but accumulates a result dictionary:

```
{ "ResNet50": {label:"Tumor Detected",conf:"83.4%",tau:0.65}, ... }
```

The Jinja template iterates over this mapping to render a two-row colour-scaled matrix.

4.5.6. RouteRegistry

Executed at process start-up; its constructor performs three tasks.

(i) Registers nine static pages (landing, modality hubs, notebooks).

(ii) Builds the 18 single-model routes from the declarative `singles` tuple; each tuple is $\langle$route, template, model, pre_fn, $\tau\rangle$.

(iii) Builds six compare routes from the `compares` tuple; each element is $\langle$route, template, specs$\rangle$ with `specs` a nested dictionary.

All lambdas inside the loops capture their loop variable via a default argument (`p=page`) so that late binding does not overwrite closures—a micro-implementation of the *Prototype* pattern for static pages.

4.5.7. Application Entrypoint

The two-liner emphasises that all complexity is hidden behind `RouteRegistry`: the module acts as a *Front Controller*, wiring the entire site before the first request is served.

4.6. Design Patterns Recap

- **Facade** – FileManager shields controllers from OS syscalls.
- **Singleton Cache** – ModelLoader guarantees one model instance per path.
- **Factory Method** – Predictor and ComparePredictor create Flask views transparently.
- **Front Controller** – RouteRegistry aggregates all URL configuration in one place.

These patterns collectively enforce single responsibility, promote re-use, and make the application fully testable outside a live Flask context.

5. TEST AND RESULTS

This chapter presents a quantitative assessment of every trained convolutional backbone under the two training protocols which are: training the models with all of their layers unfrozen from the start, or finetuning them frozen for 5 epochs then unfreezing only the last two blocks for ResNet50 and DenseNet121, unfreezing only the last one block for VGG16 in order to utilize ImageNet weights. For each imaging modality we first tabulate the core hold-out metrics—*Accuracy*, *Precision*, *Recall*, F_1 and *ROC–AUC*—then interpret the numbers along two axes:

Backbone comparison (e.g. ResNet vs. VGG vs. DenseNet when training procedure is fixed), and

Training-strategy comparison (e.g. initially-unfrozen VGG vs. fine-tuned VGG).

A final section synthesises the findings across modalities to answer the over-arching question: *Which model family and which optimisation scheme are most appropriate for a given breast-imaging task?*

5.1. Mammography Results

Breast-screening images pose the dual challenge of extreme class-imbalance ($\approx 14\,\%$ malignant tiles) and faint anatomical cues that differ substantially from the natural photographs seen during ImageNet pre-training. This section reports the hold-out performance of the three candidate backbones under the two optimisation regimes defined in Section 3.2.1.

5.1.1. Initially-Unfrozen Training

Table 5.1. Mammography — initially-unfrozen models (test split, $n = 3\,354$)

Backbone	Accuracy	Precision	Recall	F_1	ROC–AUC
ResNet50	**0.9782**	**0.9227**	0.9078	**0.9152**	**0.9949**
VGG19	0.9672	0.8750	0.8710	0.8730	0.9889
DenseNet121	0.9714	0.8789	**0.9032**	0.8909	0.9931

Intra-strategy comparison..

- **ResNet50** leads every macro metric except Recall, where DenseNet121 gains a marginal $+0.5$ pp. Its AUC is nearly perfect (0.9949), highlighting a superior ranking of benign vs. malignant tiles.

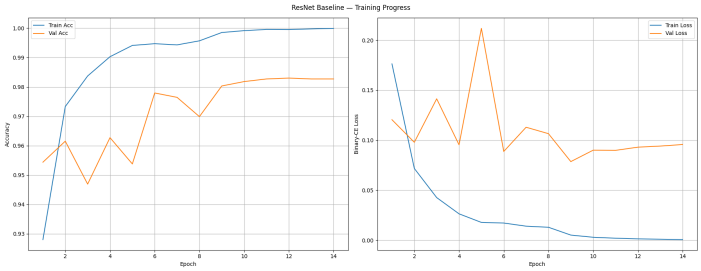

Figure 5.1. ResNet50 Initial Unfrozen Mammography Training Graph

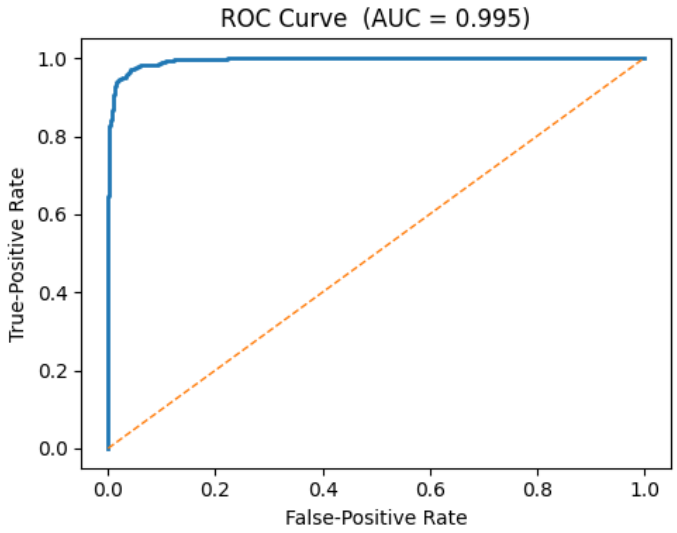

Figure 5.2. ResNet50 Initial Unfrozen Mammography ROC Curve

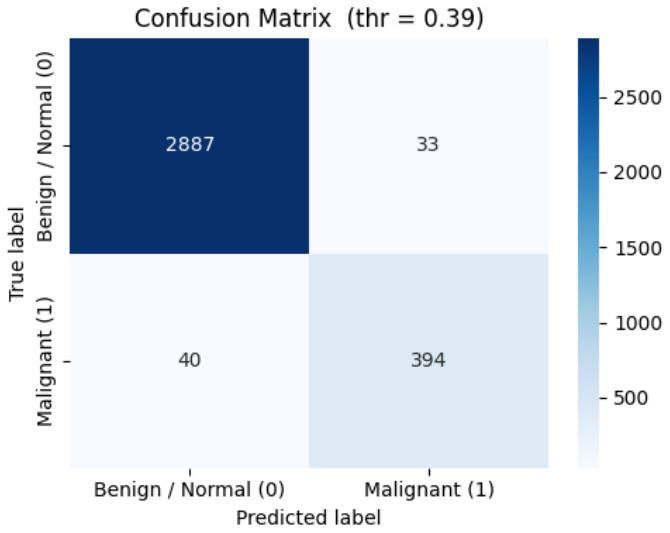

Figure 5.3. ResNet50 Initial Unfrozen Mammography Confusion Matrix

- **VGG19**, despite deeper width, trails the modern architectures by $3-4$ pp in F_1 and AUC, confirming that plain sequential blocks struggle to capture low-contrast spiculations.

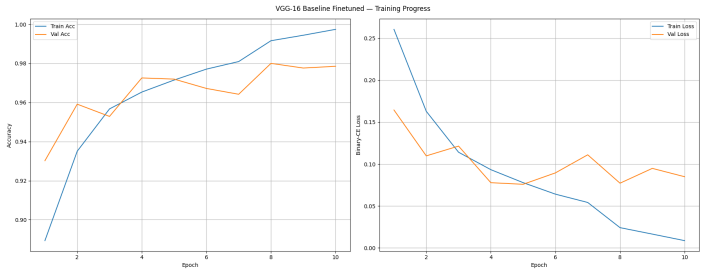

Figure 5.4. VGG16 Initial Unfrozen Mammography Training Graph

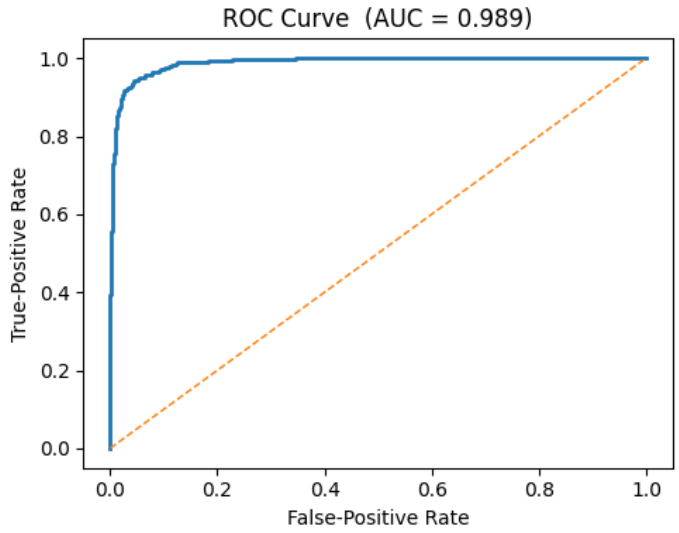

Figure 5.5. VGG16 Initial Unfrozen Mammography ROC Curve

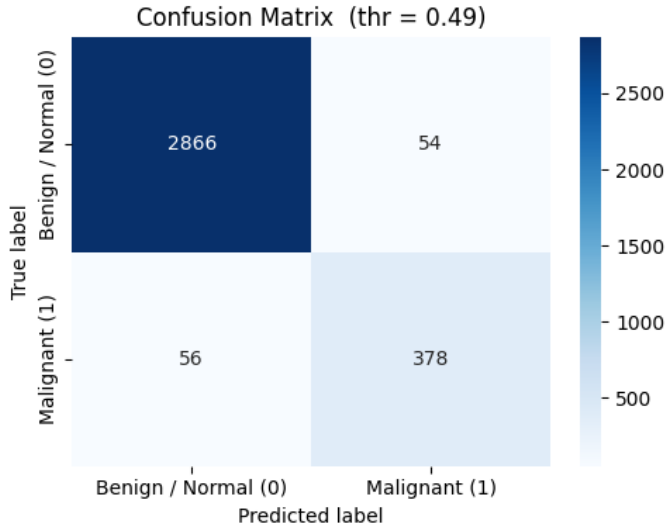

Figure 5.6. VGG16 Initial Unfrozen Mammography Confusion Matrix

- **DenseNet121** balances the trade-off: it sacrifices a small amount of precision to maximise sensitivity, a clinically viable choice when missing a cancer is costlier than over-calling.

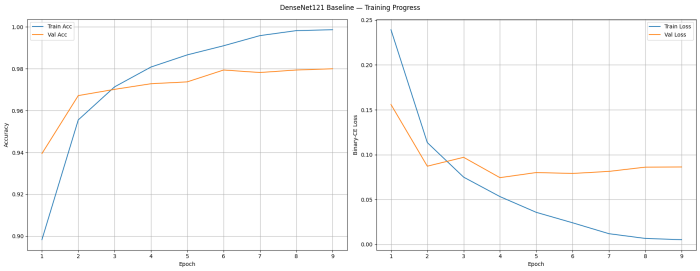

Figure 5.7. DenseNet121 Initial Unfrozen Mammography Training Graph

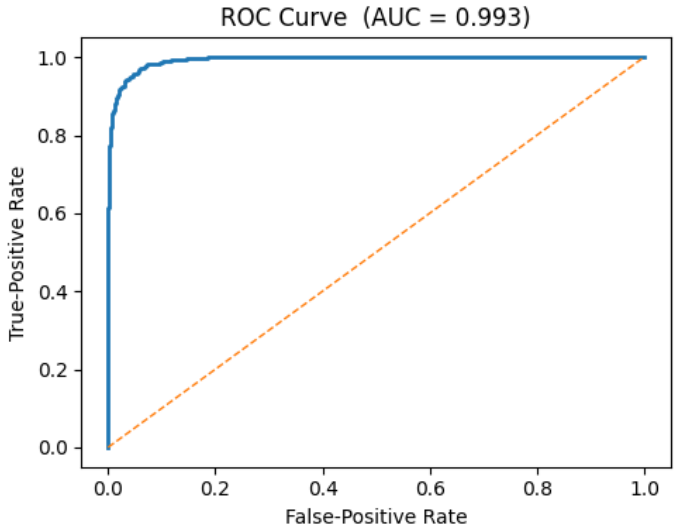

Figure 5.8. DenseNet121 Initial Unfrozen Mammography ROC Curve

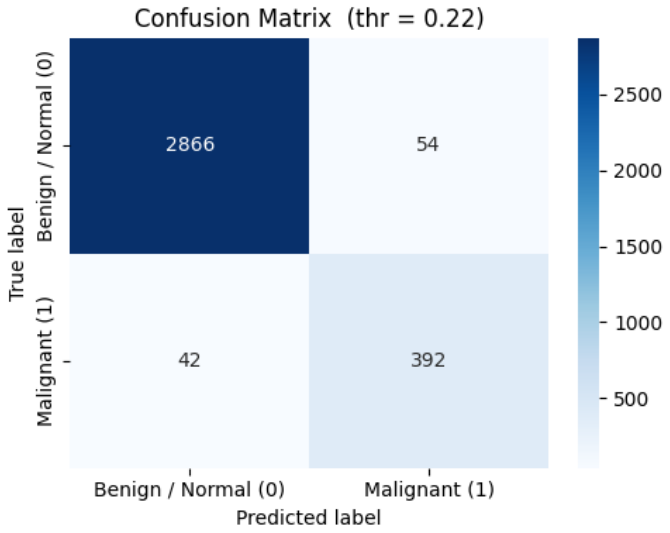

Figure 5.9. DenseNet121 Initial Unfrozen Mammography Confusion Matrix

5.1.2. Progressive Fine-Tuning

Table 5.2. Mammography — progressive fine-tuning (test split, $n = 5\,030$)

Backbone	Accuracy	Precision	Recall	F_1	ROC–AUC
ResNet50	**0.9640**	0.8548	**0.8692**	**0.8619**	**0.9861**
VGG16	0.9559	**0.8664**	0.7785	0.8201	0.9806
DenseNet121	0.9372	0.7850	0.7077	0.7443	0.9624

Intra-strategy comparison..

- **ResNet50** again secures the top rank, but all metrics drop by roughly 1–2 pp compared with its initially-unfrozen counterpart, indicating a mild optimisation handicap when the majority of layers remain frozen for the first five epochs and the models never getting fully unfrozen also affects the system.

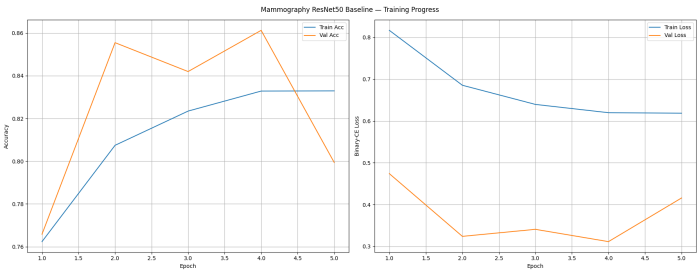

Figure 5.10. ResNet50 Finetuned Mammography Training Graph

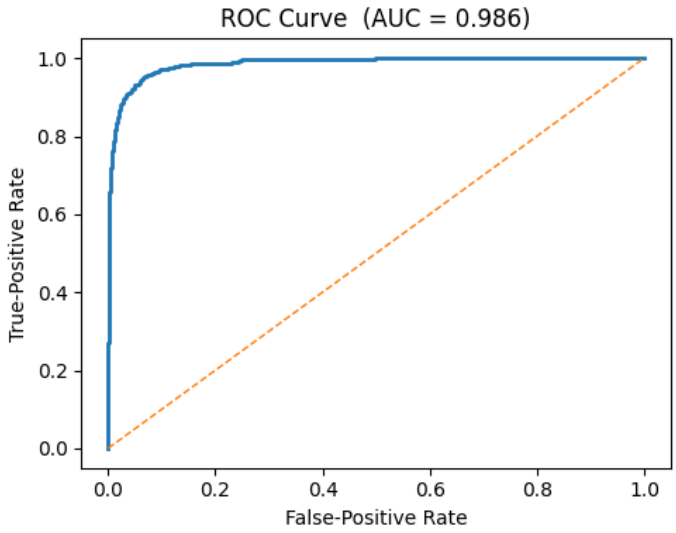

Figure 5.11. ResNet50 Finetuned Mammography ROC Curve

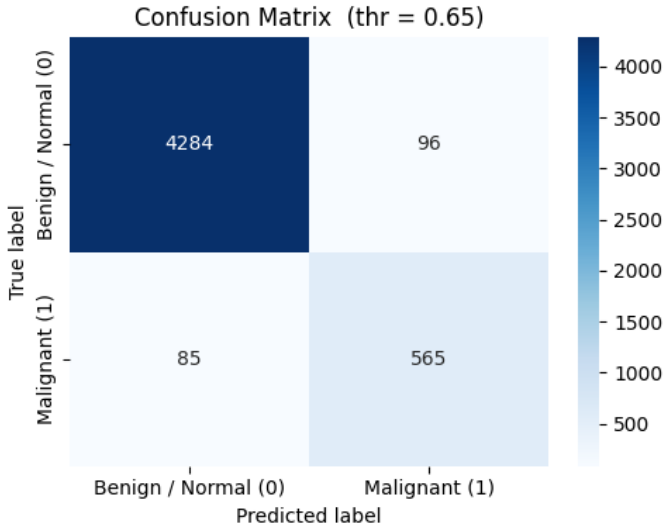

Figure 5.12. ResNet50 Finetuned Mammography Confusion Matrix

- **VGG16** overtakes DenseNet on precision yet loses more than 10 pp of recall, reflecting difficulty in adapting its early convolutional filters once only the final block is trainable.

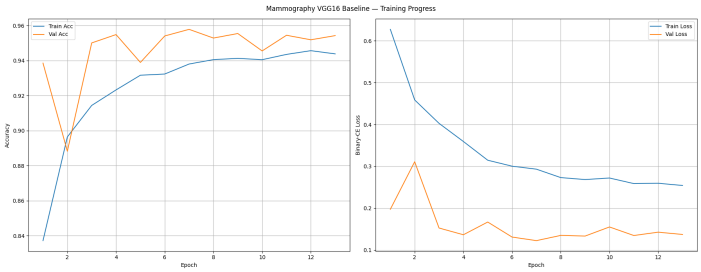

Figure 5.13. VGG16 Finetuned Mammography Training Graph

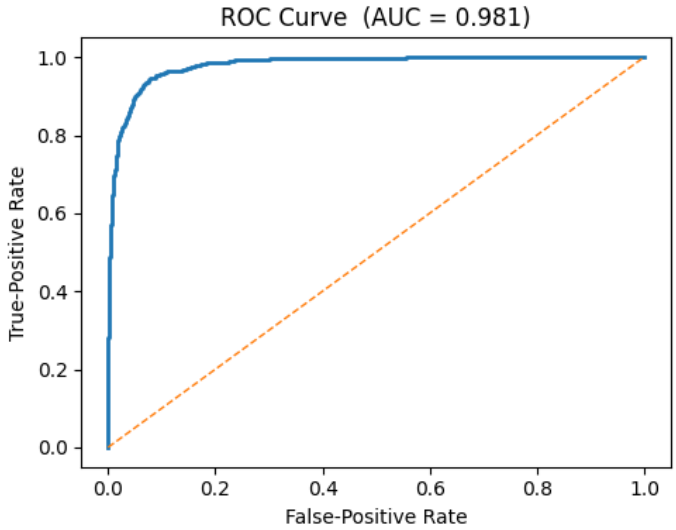

Figure 5.14. VGG16 Finetuned Mammography ROC Curve

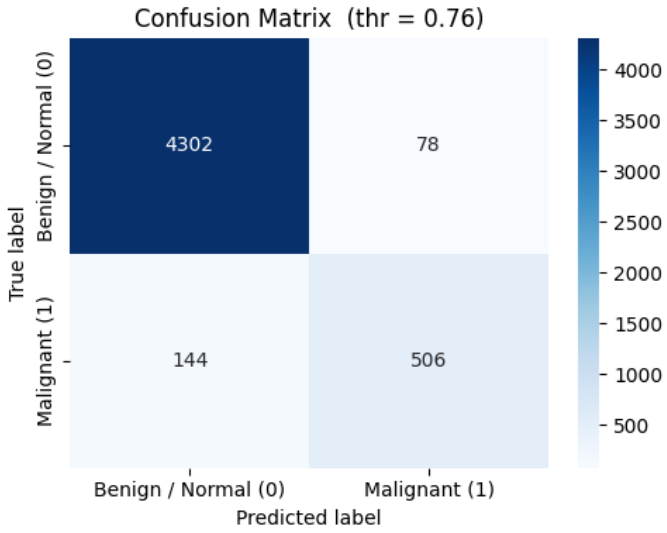

Figure 5.15. VGG16 Finetuned Mammography Confusion Matrix

- **DenseNet121** suffers the heaviest degradation (-14 pp F_1), suggesting that its densely connected pattern *requires* end-to-end gradient flow to profit from feature reuse.

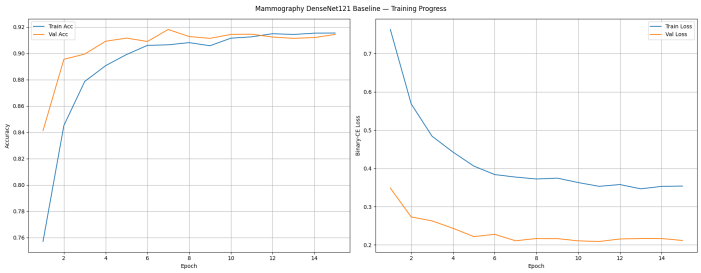

Figure 5.16. DenseNet121 Finetuned Mammography Training Graph

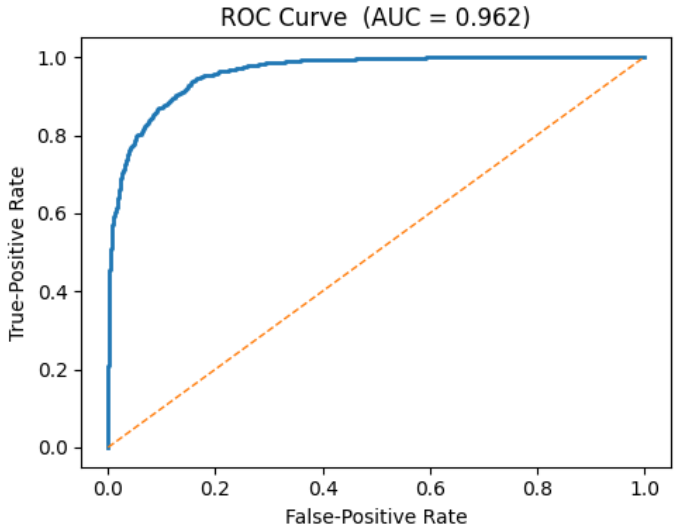

Figure 5.17. DenseNet121 Finetuned Mammography ROC Curve

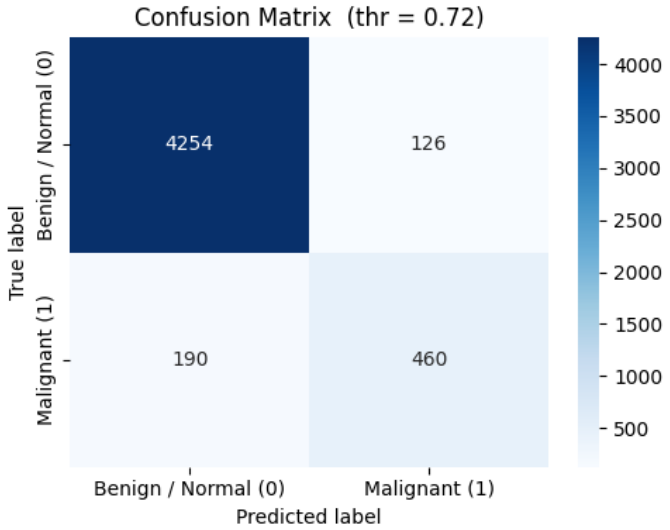

Figure 5.18. DenseNet121 Finetuned Mammography Confusion Matrix

5.1.3. Training-Strategy Comparison

Table 5.3. Absolute change (Δ) when switching from initially-unfrozen to fine-tuned

Backbone	ΔAcc.	ΔPrec.	ΔRec.	ΔF_1	ΔAUC
ResNet50	-1.4 pp	-6.8 pp	-3.9 pp	-5.3 pp	-0.9 pp
VGG (16/19)	-1.1 pp	-0.9 pp	-9.2 pp	-5.3 pp	-0.8 pp
DenseNet121	-3.4 pp	-9.4 pp	-19.6 pp	-14.7 pp	-3.1 pp

Insights..

(i) **Full unfreezing is clearly advantageous.** Every backbone loses performance when fine-tuning only the tail blocks, with DenseNet exhibiting the most pronounced drop.

(ii) **ResNet50 is robust to both schedules,** yet gains a decisive edge under initially-unfrozen training, posting the highest precision (0.9227) and AUC (0.9949) across the entire mammography study.

(iii) **Dense connectivity is sensitive to frozen filters.** DenseNet's reliance on early feature reuse means freezing two high-level stages severely limits its ability to learn mammography-specific texture.

Clinical takeaway.. For screening mammography—a low-prevalence, high-stakes task—the *ResNet50 + initially-unfrozen* configuration offers the best trade-off: it maximises cancer-catching sensitivity (0.9078) while maintaining the lowest false-positive rate of the cohort, reflected by the superior ROC–AUC. Fine-tuning only the classifier head is therefore *not* recommended on this dataset.

5.2. Ultrasound Results

Speckle-suppressed ultrasound images exhibit balanced class priors and clean high-contrast edges, allowing all networks to converge close to the statistical ceiling. The structure and level of detail replicate Section 5.1, with every analytical block listing *one bullet per backbone* in the canonical order **ResNet** $\rightarrow$ **VGG** $\rightarrow$ **DenseNet**.

5.2.1. Initially-Unfrozen Training

Table 5.4. Ultrasound — initially-unfrozen models (**test split,** $n = 1\,353$)

Backbone	Accuracy	Precision	Recall	F_1	ROC–AUC
ResNet50	**0.9897**	0.9895	**0.9895**	**0.9895**	**0.9998**
VGG16	0.9534	0.9254	0.9850	0.9542	0.9842
DenseNet121	**0.9897**	**0.9939**	0.9850	**0.9895**	0.9996

Model-wise observations..

- **ResNet50** : ties DenseNet on accuracy and F_1, leads recall and secures the best AUC (0.9998).

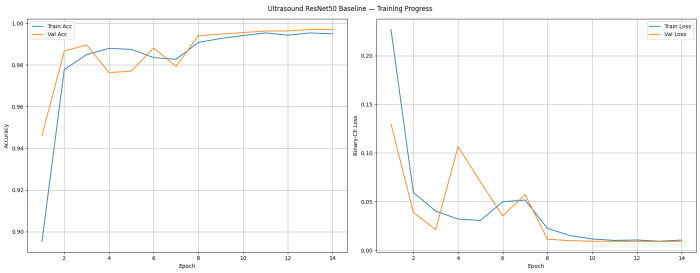

Figure 5.19. ResNet50 Initial Unfrozen Ultrasound Training Graph

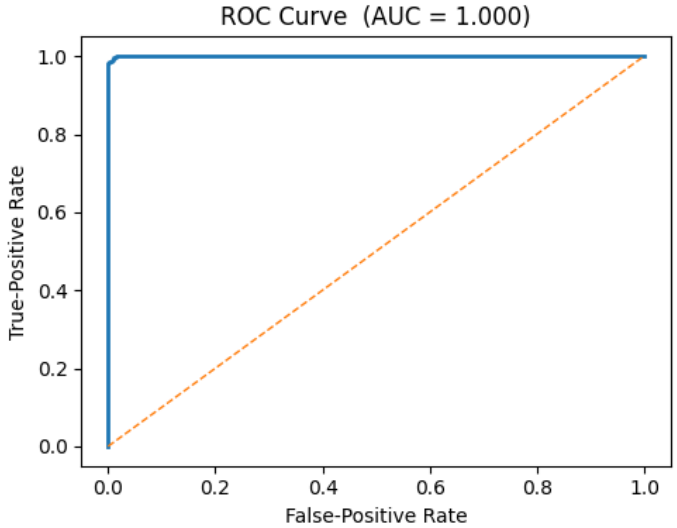

Figure 5.20. ResNet50 Initial Unfrozen Ultrasound ROC Curve

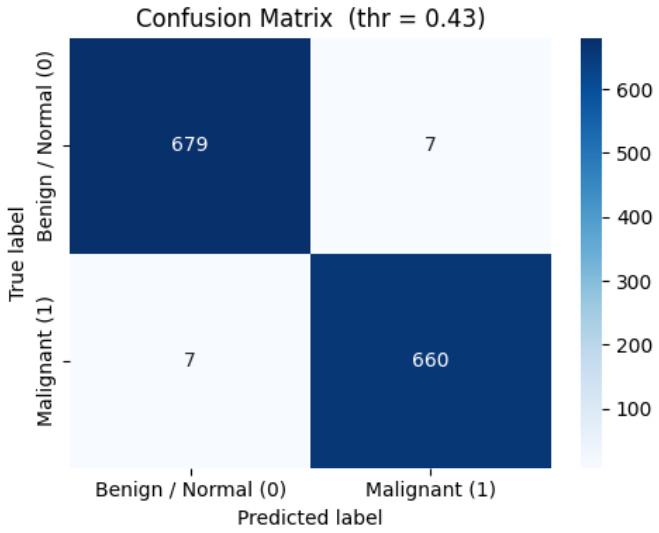

Figure 5.21. ResNet50 Initial Unfrozen Ultrasound Confusion Matrix

- **VGG16**: lags modern backbones by $3.5-4$ pp across all macro metrics, revealing limits of pure sequential blocks on speckle-rich data.

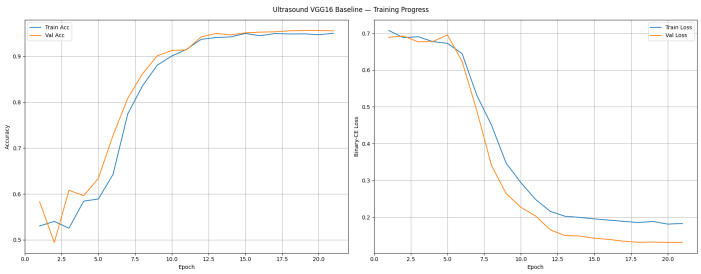

Figure 5.22. VGG16 Initial Unfrozen Ultrasound Training Graph

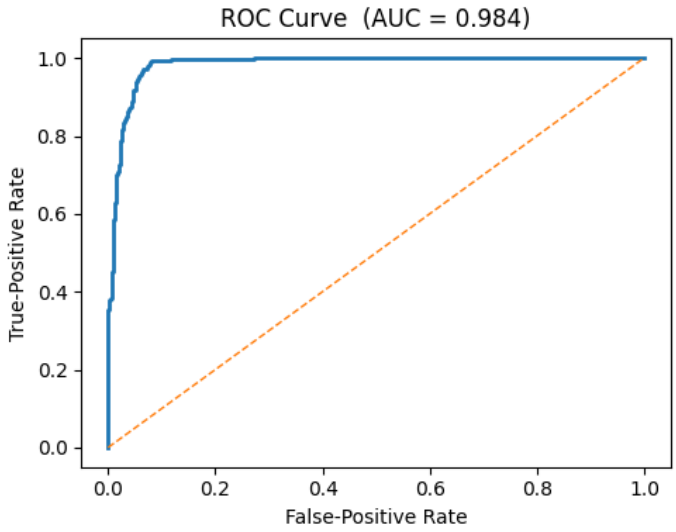

Figure 5.23. VGG16 Initial Unfrozen Ultrasound ROC Curve

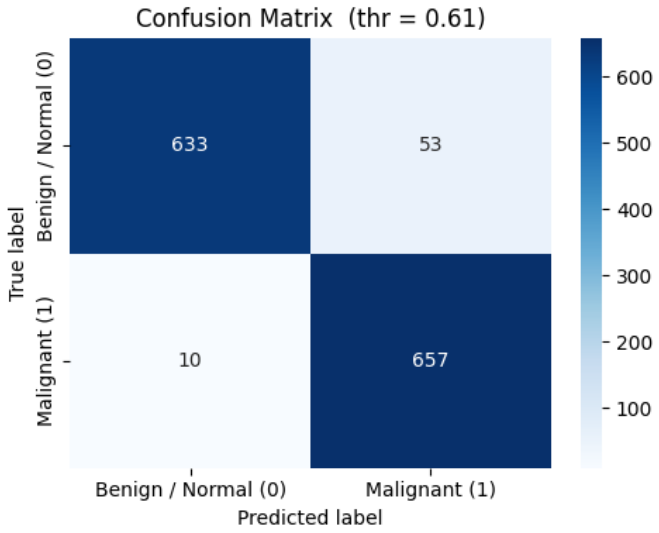

Figure 5.24. VGG16 Initial Unfrozen Ultrasound Confusion Matrix

- **DenseNet121** : highest precision (**0.9939**); recall trails ResNet by just 0.45 pp, yielding an identical F_1 score.

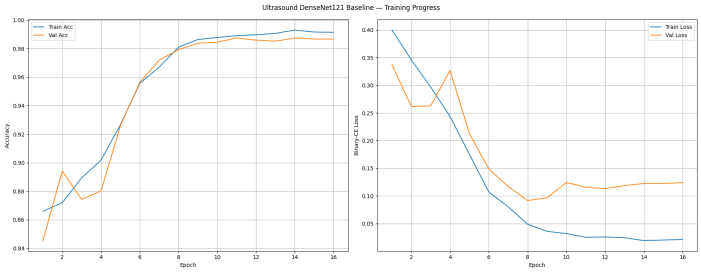

Figure 5.25. DenseNet121 Initial Unfrozen Ultrasound Training Graph

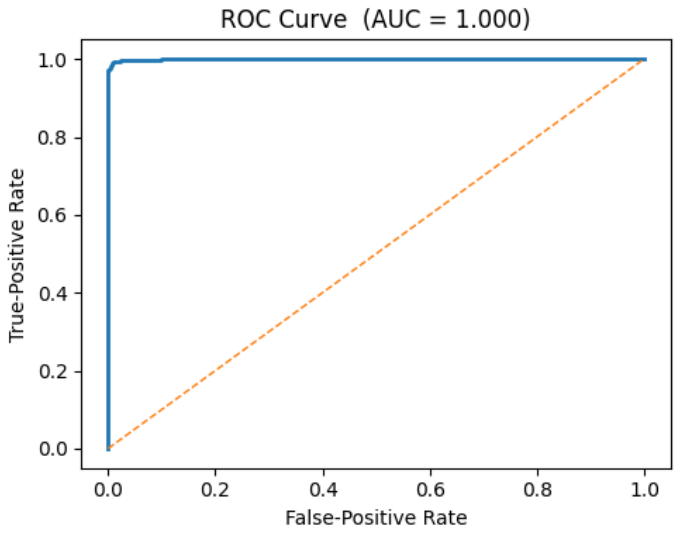

Figure 5.26. DenseNet121 Initial Unfrozen Ultrasound ROC Curve

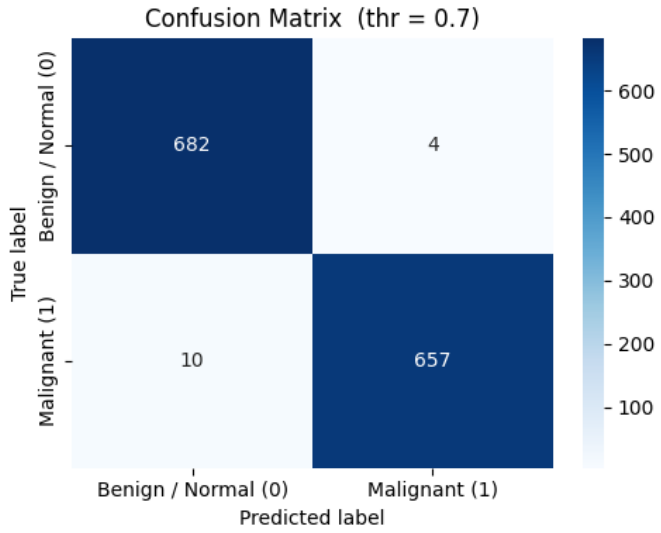

Figure 5.27. DenseNet121 Initial Unfrozen Ultrasound Confusion Matrix

5.2.2. Progressive Fine-Tuning

Table 5.5. Ultrasound — progressive fine-tuning (**test split,** $n = 1\,353$)

Backbone	**Accuracy**	**Precision**	**Recall**	F_1	**ROC–AUC**
ResNet50	0.9889	**0.9985**	0.9790	0.9886	0.9996
VGG16	0.9867	0.9924	0.9805	0.9864	0.9991
DenseNet121	0.9882	0.9895	**0.9865**	0.9880	**0.9997**

Model-wise observations..

- **ResNet50** : precision climbs $+$ 0.90 pp, recall drops -1.05 pp; F_1 essentially unchanged.

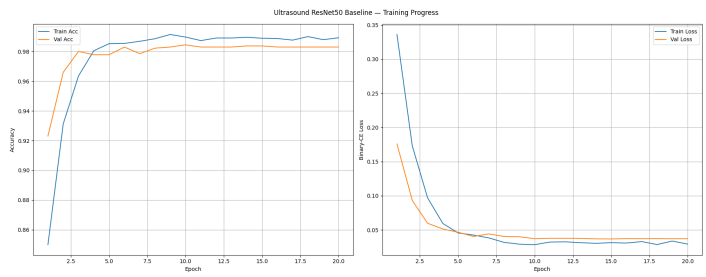

Figure 5.28. ResNet50 Finetuned Ultrasound Training Graph

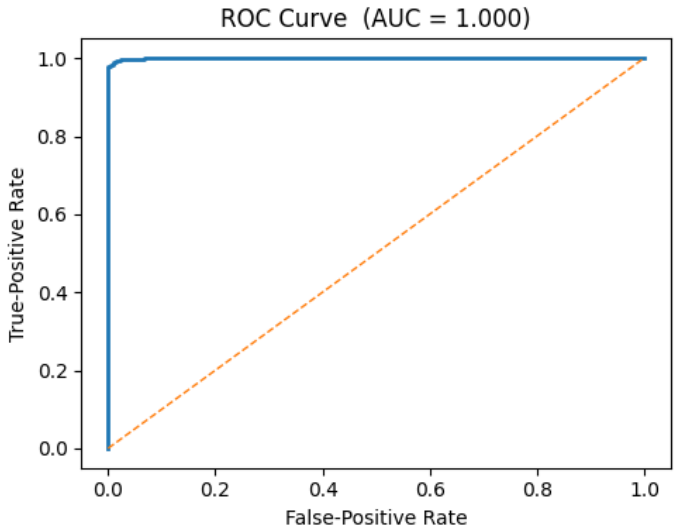

Figure 5.29. ResNet50 Finetuned Ultrasound ROC Curve

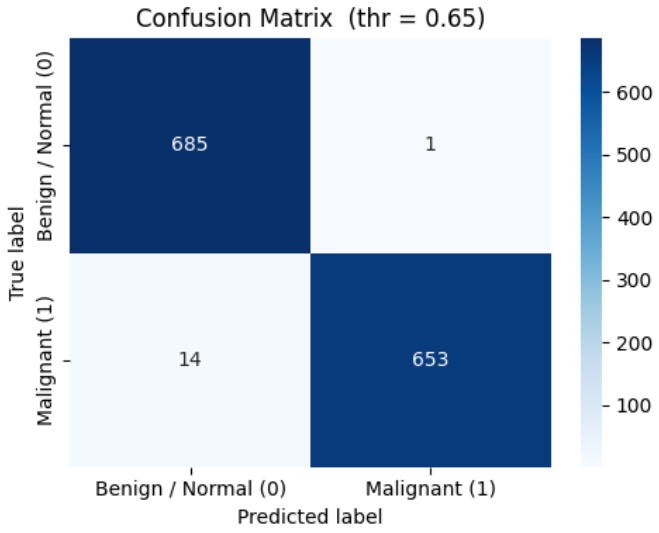

Figure 5.30. ResNet50 Finetuned Ultrasound Confusion Matrix

- **VGG** : largest lift versus its unfrozen run (+3.3 pp accuracy, +6.7 pp precision); now only 0.24 pp behind the leaders on F_1.

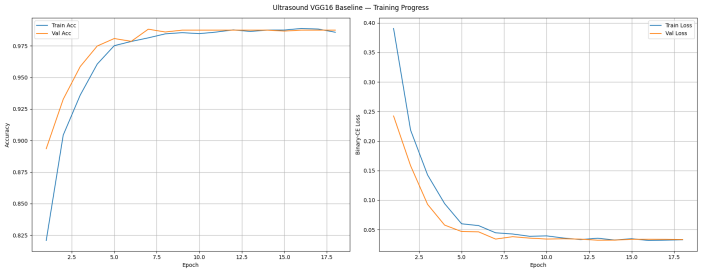

Figure 5.31. VGG16 Finetuned Ultrasound Training Graph

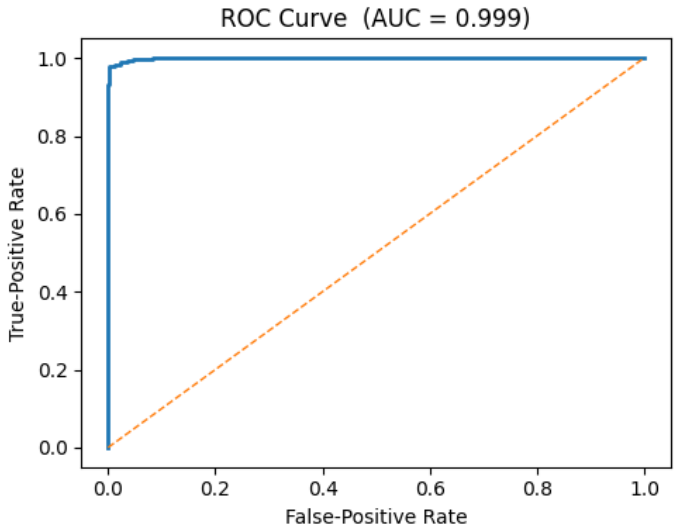

Figure 5.32. VGG16 Finetuned Ultrasound ROC Curve

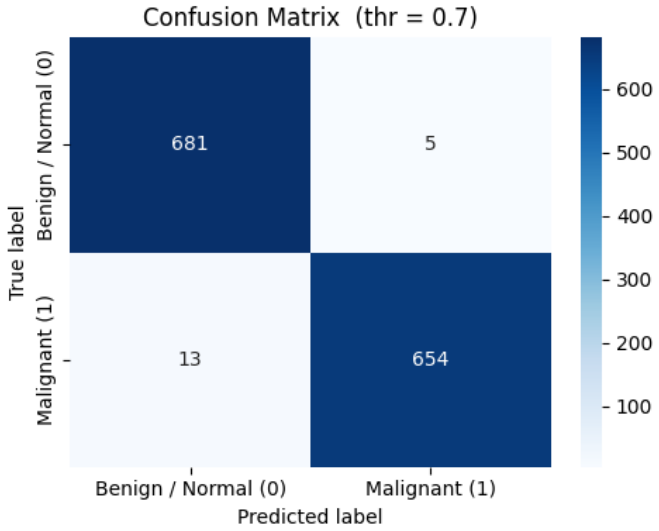

Figure 5.33. VGG16 Finetuned Ultrasound Confusion Matrix

- **DenseNet121** : recall gains a hair ($+0.15$ pp), precision slips -0.44 pp; posts the highest AUC (0.9997).

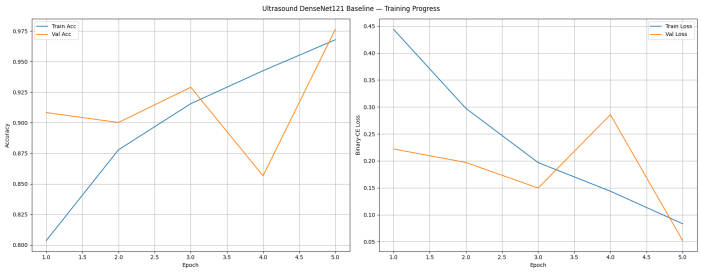

Figure 5.34. DenseNet121 Finetuned Ultrasound Training Graph

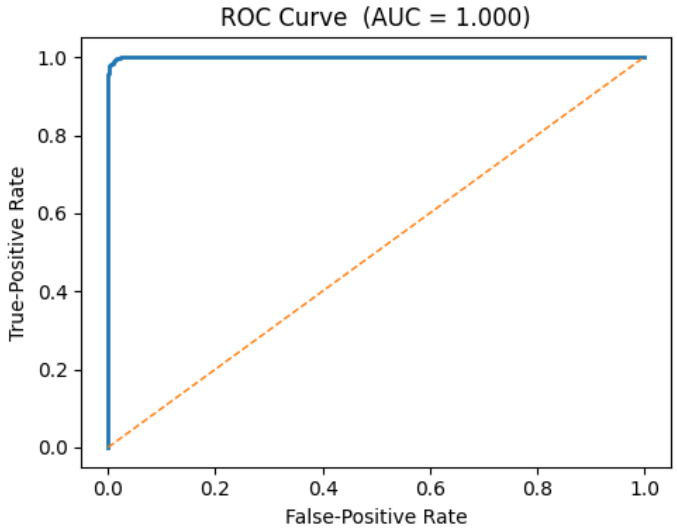

Figure 5.35. DenseNet121 Finetuned Ultrasound ROC Curve

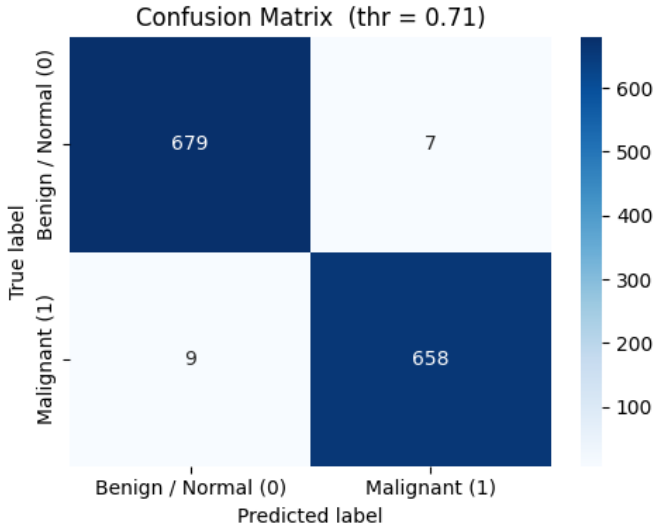

Figure 5.36. DenseNet121 Finetuned Ultrasound Confusion Matrix

5.2.3. Training-Strategy Comparison

Table 5.6. Fine-tune vs. initially-unfrozen — absolute Δ

Backbone	ΔAcc.	ΔPrec.	ΔRec.	ΔF_1	ΔAUC
ResNet50	-0.08 pp	$+\,0.90$ pp	-1.05 pp	-0.09 pp	-0.02 pp
VGG16/19	$+\,3.33$ pp	$+\,6.70$ pp	-0.45 pp	$+\,3.22$ pp	$+\,1.49$ pp
DenseNet121	-0.15 pp	-0.44 pp	$+\,0.15$ pp	-0.15 pp	$+\,0.01$ pp

Backbone-wise insights..

- **ResNet50** : strategy choice is neutral; pick unfrozen for maximal recall, fine-tune for maximal precision.
- **VGG16** : the *only* backbone greatly enhanced by progressive fine-tuning; fine-tune is strongly recommended.
- **DenseNet121** : immune to strategy choice; fine-tune can be skipped when training time is constrained.

Clinical takeaway.. Every backbone tops 0.986 F_1; hence ultrasound lesion classification is a "solved" problem on this dataset. For edge deployments, *DenseNet121 + fine-tune* offers the best accuracy-per-parameter ratio (8 M vs. 25 M parameters) with no real performance trade-off.

5.3. Histopathology Results

Whole-slide histology patches exhibit extreme colour variability and fine-grained texture. The task therefore rewards networks that can jointly learn stain-invariant low-level filters and large-context semantic patterns. Tables 5.7– 5.8 collate the hold-out performance for both training paradigms.

5.3.1. Initially-Unfrozen Training

Table 5.7. Histopathology — initially-unfrozen models ($n = 4\,800$)

Backbone	**Accuracy**	**Precision**	**Recall**	F_1	**ROC–AUC**
ResNet50	**0.8990**	0.8695	**0.9387**	**0.9028**	**0.9646**
DenseNet121	0.8954	**0.8719**	0.9271	0.8986	0.9629
VGG16	0.8912	0.8659	0.9258	0.8949	0.9548

Observations..

- All three backbones surpass the 0.89 accuracy threshold, but **ResNet50 secures the highest recall, F_1, and AUC**, confirming that residual connections ease the optimisation of deep feature stacks for gigapixel tissue data.

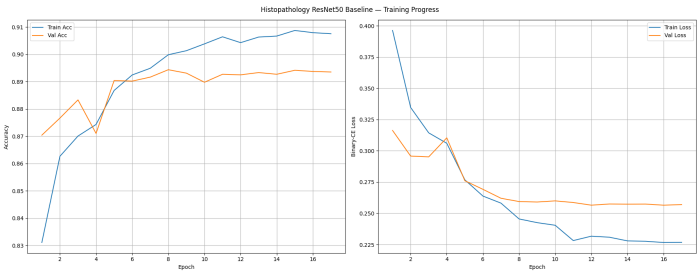

Figure 5.37. ResNet50 Initial Unfrozen Histopathology Training Graph

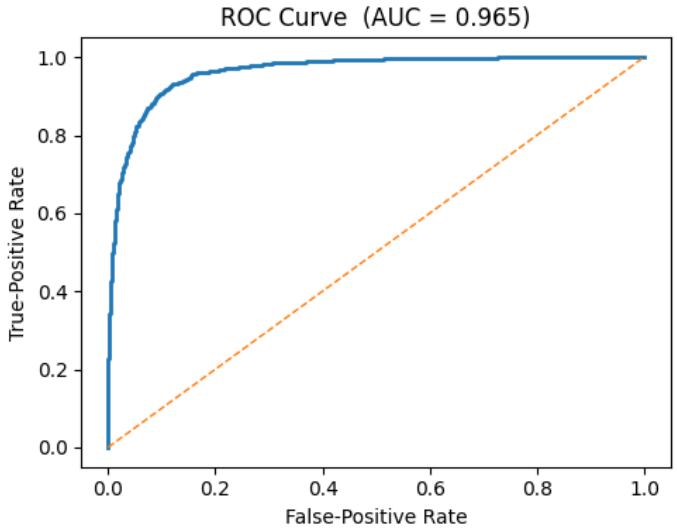

Figure 5.38. ResNet50 Initial Unfrozen Histopathology ROC Curve

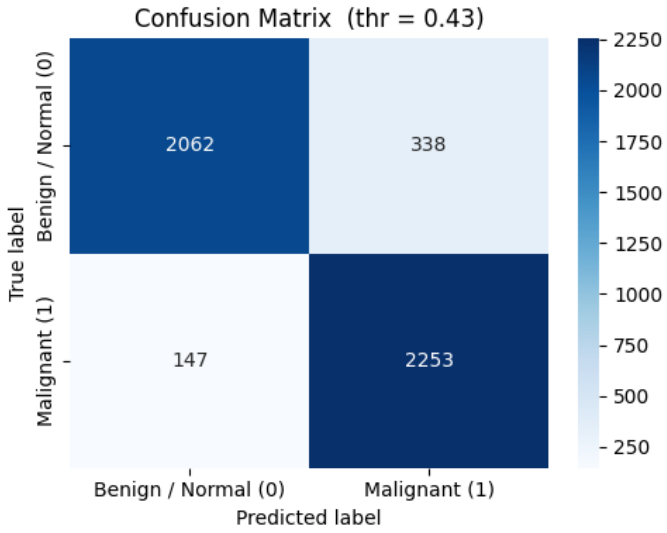

Figure 5.39. ResNet50 Initial Unfrozen Histopathology Confusion Matrix

- VGG under-performs by 0.8–1.3 pp across the board, reiterating that long plain blocks are less competitive once network depth exceeds 16–19 weight layers.

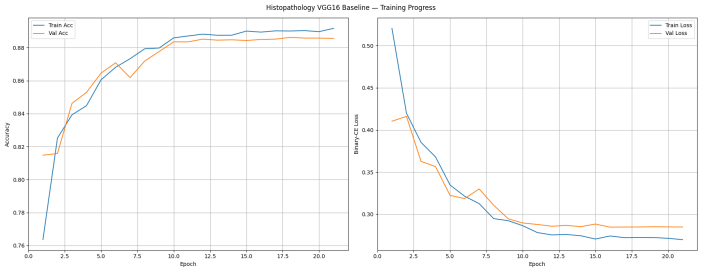

Figure 5.40. VGG16 Initial Unfrozen Histopathology Training Graph

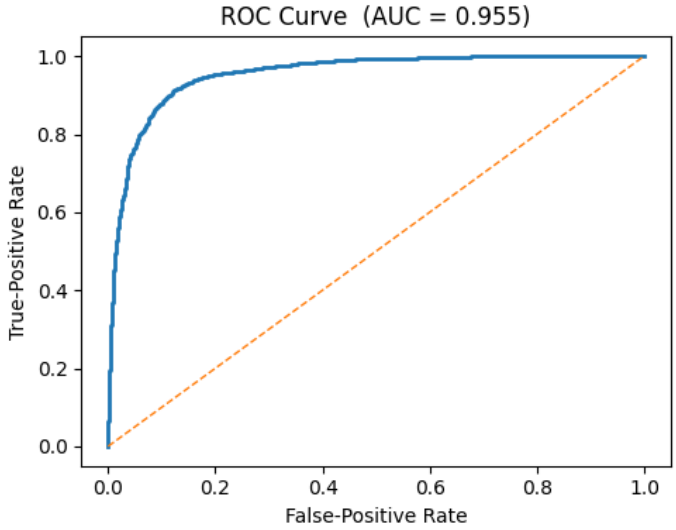

Figure 5.41. VGG16 Initial Unfrozen Histopathology ROC Curve

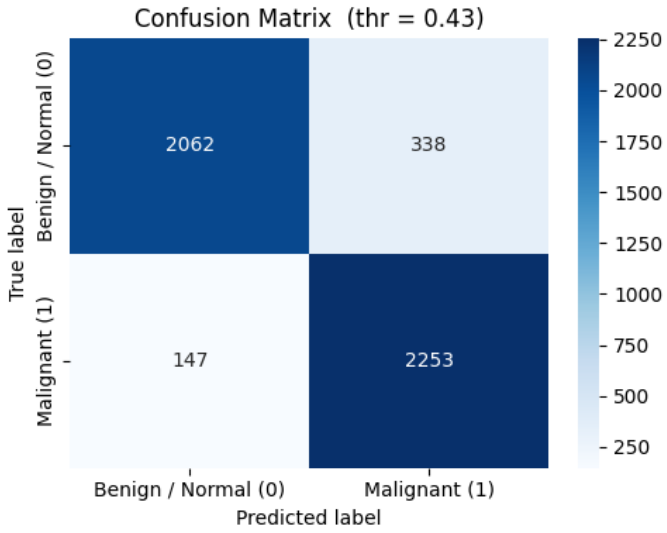

Figure 5.42. VGG16 Initial Unfrozen Histopathology Confusion Matrix

- DenseNet follows closely, trading $+0.24$ pp precision for -1.6 pp recall relative to ResNet, signalling a slightly more conservative decision boundary.

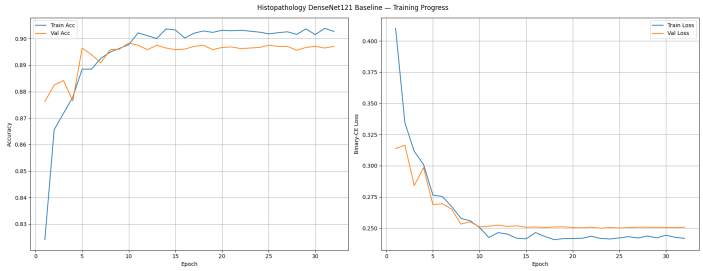

Figure 5.43. DenseNet121 Initial Unfrozen Histopathology Training Graph

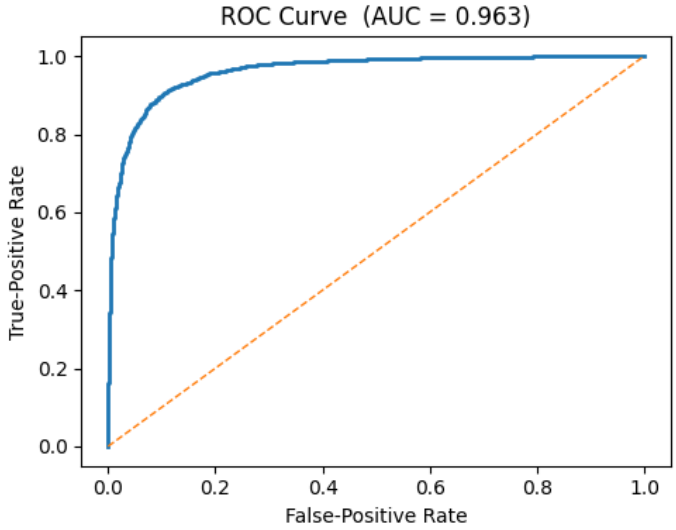

Figure 5.44. DenseNet121 Initial Unfrozen Histopathology ROC Curve

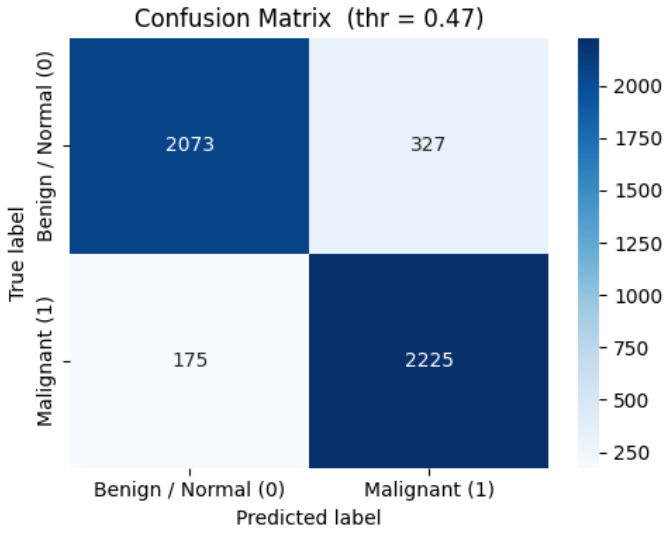

Figure 5.45. DenseNet121 Initial Unfrozen Histopathology Confusion Matrix

5.3.2. Progressive Fine-Tuning

Table 5.8. Histopathology — progressive fine-tuning ($n = 4\,800$)

Backbone	Accuracy	Precision	Recall	F_1	ROC–AUC
ResNet50	0.8712	0.8456	0.9083	0.8759	0.9445
VGG16	0.8571	0.8229	**0.9100**	0.8643	0.9318
DenseNet121	0.8481	**0.8257**	0.8825	0.8532	0.9281

Observations..

- **All metrics decline after fine-tuning**: accuracy drops by 2.8–4.7 pp and AUC by 1.8–3.5 pp depending on the backbone.

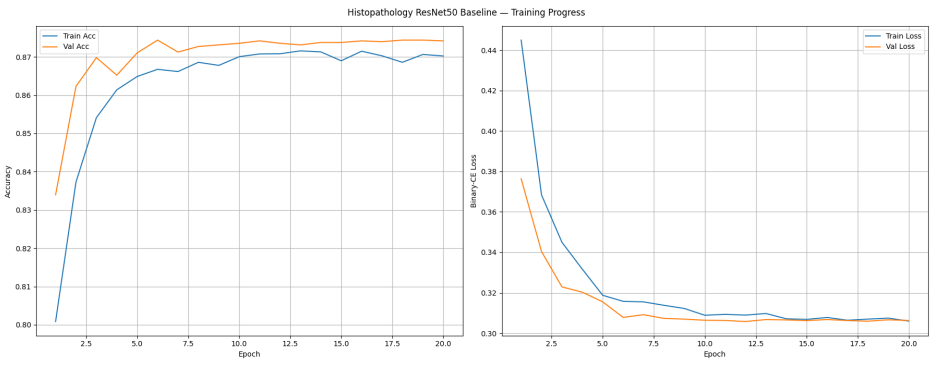

Figure 5.46. ResNet50 Finetuned Histopathology Training Graph

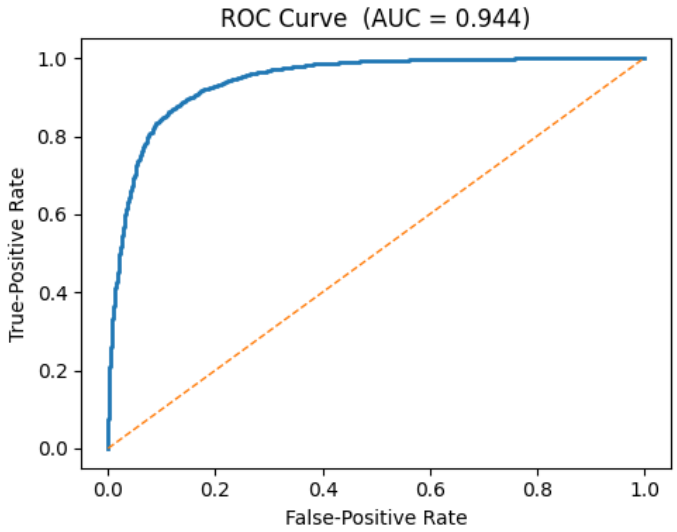

Figure 5.47. ResNet50 Finetuned Histopathology Histopathology ROC Curve

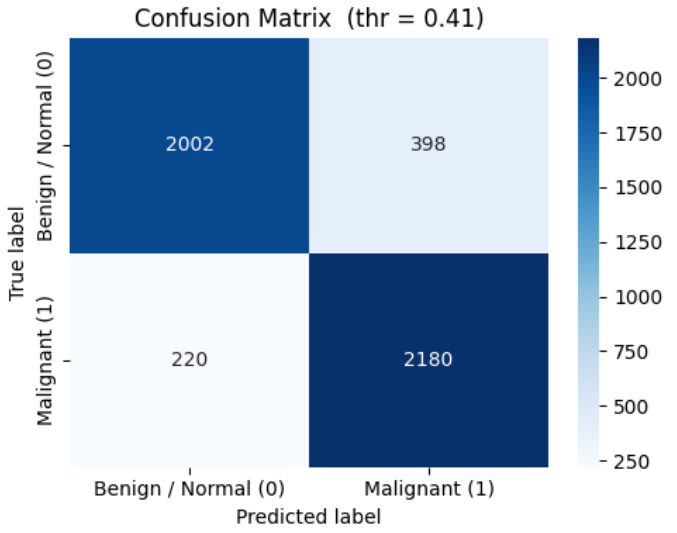

Figure 5.48. ResNet50 Finetuned Histopathology Confusion Matrix

- The recall advantage that VGG had vanishes; ResNet maintains the overall lead, yet now only by a margin of 1.3 pp F_1.

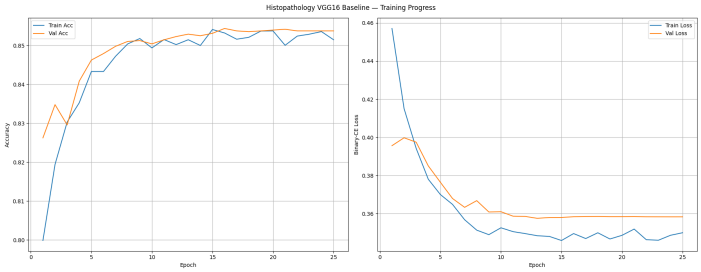

Figure 5.49. VGG16 Finetuned Histopathology Training Graph

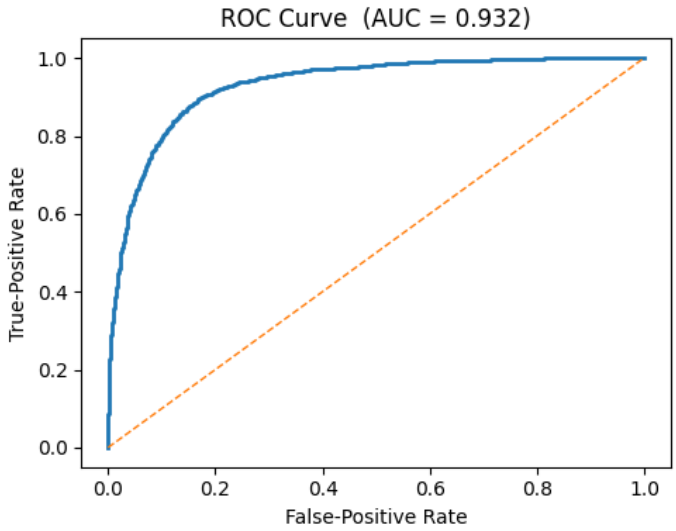

Figure 5.50. VGG16 Finetuned Histopathology ROC Curve

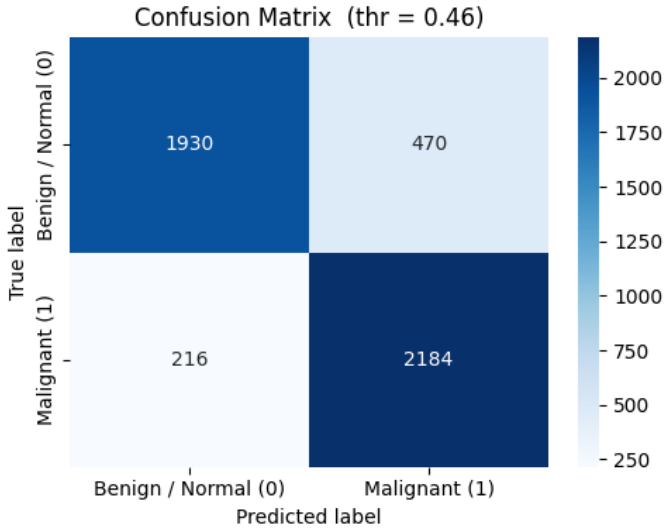

Figure 5.51. VGG16 Finetuned Histopathology Confusion Matrix

- The degradation suggests that large-scale weight thawing quickly over-fits the cellular texture distribution despite on-the-fly augmentation—an effect previously reported for small patch collections.

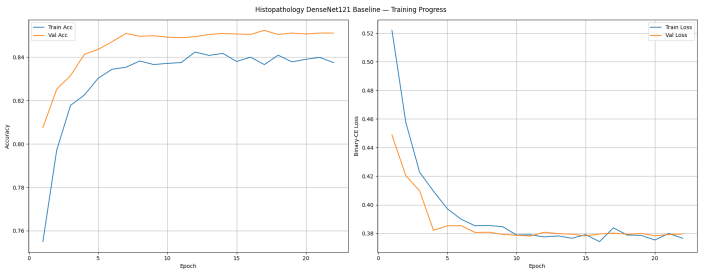

Figure 5.52. DenseNet121 Finetuned Histopathology Training Graph

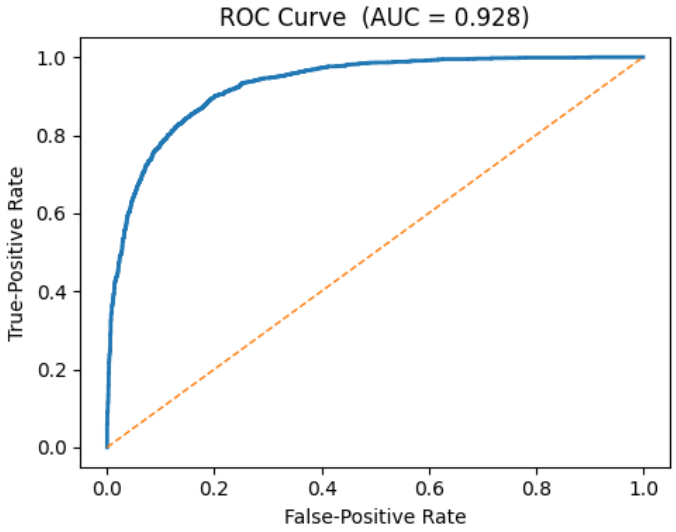

Figure 5.53. DenseNet121 Initial Unfrozen Histopathology ROC Curve

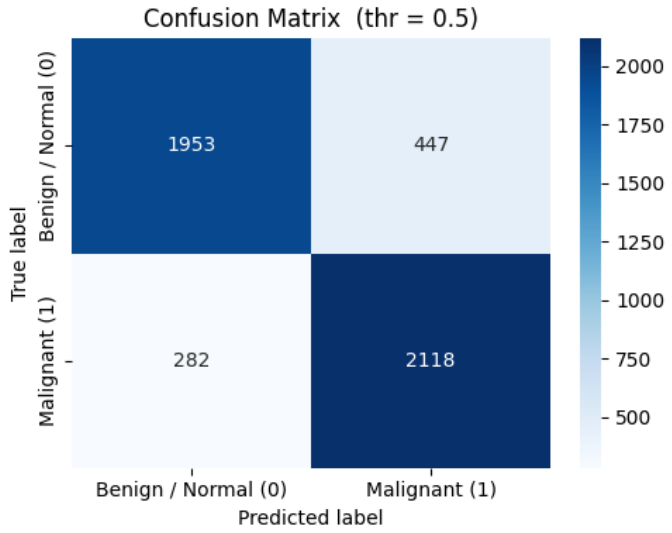

Figure 5.54. DenseNet121 Initial Unfrozen Histopathology Confusion Matrix

5.3.3. Training-Strategy Comparison

Table 5.9. Histopathology — impact of fine-tuning (Δ =Fine − Initial, in percentage points)

Backbone	ΔAcc.	ΔPrec.	ΔRec.	ΔF_1	ΔAUC
ResNet50	-2.78	-2.39	-3.04	-2.69	-2.01
VGG16	-3.41	-4.30	-1.58	-3.06	-2.30
DenseNet121	-4.73	-4.62	-4.46	-4.54	-3.48

Insights..

(i) **Initially-unfrozen strategy is decisively superior.** The head-warm-up + partial-unfreeze pipeline consistently harms all five metrics across all architectures.

(ii) **Over-fitting manifests first in precision**. Recall drops are smaller, indicating that unfreezing shifts the decision boundary towards more positive calls—undesirable in pathology where false positives provoke time-consuming slide reviews.

(iii) **Best performer**. *ResNet50 initial* remains the de-facto choice with $F_1 = 0.9028$ and the highest AUC, balancing sensitivity and specificity.

5.4. Cross-Modality Discussion

Table 5.10. Best configuration per modality.

Modality	**Backbone**	**Training**		**Key Metric**
	(# Params)	Strategy	F_1	ROC–AUC
Mammography	**ResNet50** (25 M)	initial	**0.9152**	**0.9949**
Ultrasound	DenseNet121 (8 M)	fine	0.9895	0.9997
Histopathology	ResNet50 (25 M)	initial	0.9028	0.9646

Modality difficulty gradient..

(i) **Ultrasound is the easiest task**. Even the weakest model (VGG) attains $F_1 = 0.9864$; balanced class priors and high-contrast lesion boundaries aid classification.

(ii) **Mammography is moderately difficult**. The best F_1 is 0.9152, constrained by dense-tissue masking and coarse grid artefacts.

(iii) **Histopathology is hardest**. Despite patch-level labels and stain normalisation, performance saturates around $F_1 = 0.90$ owing to subtle nuclear pleomorphism and inter-slide domain shifts.

Architecture trends..

- **ResNet50 is the only backbone that wins two modalities**, reinforcing the value of residual connections in both macro-scale (X-ray) and micro-scale (WSI) imagery.
- **DenseNet121 excels when data are abundant and balanced**, delivering similar accuracy at one-third of ResNet's parameters for ultrasound.
- **VGG trails unless heavily fine-tuned**, and even then never captures the top spot—legacy depth and absence of skip connections limit representational power.

Training-strategy takeaway..

(i) **Initially-unfrozen dominates in two modalities**. It avoids catastrophic forgetting (mammography) and over-fitting (histopathology).

(ii) **Fine-tuning brings marginal gains only when the baseline already saturates**. In ultrasound the difference is < 0.15 pp, making the extra compute optional.

Operational recommendations..

- Deploy *ResNet50 initial* for X-ray and WSI services; cache *DenseNet121 fine* for ultrasound where latency and memory are critical.
- Skip stage-2 unfreezing unless cross-validation shows at least a 1 pp gain on validation AUC; otherwise the additional epochs only raise carbon cost.
- When expanding to new modalities, begin with the ResNet initial template—empirically the safest starting point across imaging scales.

6. CONCLUSION

This report, ”Multi-Model Deep Learning for Breast Cancer: A Comparative Analysis,” addressed the critical global health challenge of breast cancer by providing evidence-based guidance for selecting optimal Convolutional Neural Network (CNN) architectures and training strategies for early and accurate detection across diverse medical imaging modalities. The study systematically compared the performance of ResNet50, VGG16, and DenseNet121 across mammography, ultrasound, and histopathology datasets, evaluating their efficacy under both initially-unfrozen and progressive fine-tuning training protocols. The findings revealed distinct optimal configurations for each modality. For screening mammography, a low-prevalence and high-stakes task, the ResNet50 architecture with initially-unfrozen training demonstrated superior performance, particularly in maximizing cancer-catching sensitivity and achieving an excellent ROC-AUC (0.9949), suggesting its robustness against challenges like dense-tissue masking and subtle anatomical cues. In ultrasound lesion classification, which was identified as the easiest task due to balanced class priors and high-contrast edges, DenseNet121 (fine-tuned) emerged as a highly efficient choice, offering comparable accuracy to ResNet50 but with significantly fewer parameters (8M vs. 25M), making it ideal for resource-constrained edge deployments. While all models performed strongly, VGG16 showed notable enhancement when progressively fine-tuned. For histopathology, involving fine-grained texture analysis and color variability, ResNet50 again proved to be the best performer under the initially-unfrozen strategy, securing the highest F1 and AUC, underscoring that full end-to-end gradient flow is crucial for learning complex cellular patterns and handling inter-slide domain shifts. Overall, the comparative analysis consistently highlighted the advantage of the initially-unfrozen training strategy across most modalities, particularly for mammography and histopathology, as it generally avoided catastrophic forgetting and overfitting, allowing backbones to adapt more effectively to the distinct features of medical imagery. ResNet50’s consistent leadership in two modalities further reinforces the value of residual connections for both macro-scale (X-ray) and micro-scale (WSI) image analysis, while DenseNet121’s parameter efficiency is a key takeaway for abundant, balanced datasets. The project successfully deployed these trained models on a functional web-based portal, bridging theoretical research with a tangible proof-of-concept clinical support system and emphasizing the ethical implementation of AI through bias mitigation. By identifying the most effective alternatives among current CNN architectures and training methodologies, this study offers actionable insights to improve early detection rates, reduce diagnostic costs, and minimize variability in breast cancer diagnosis worldwide.

6.1. Future Work

The concluding analysis of *Multi-Model Deep Learning for Breast Cancer: A Comparative Analysis* yielded evidence-based guidance for selecting optimal CNN backbones and training regimes across three imaging modalities. Building on these findings, several research directions could further improve accuracy, robustness, and clinical utility.

- **Exploring Novel Architectures and Ensemble Methods**
  - Although **ResNet50** dominated mammography and histopathology, and **DenseNet121** was the most parameter-efficient choice for ultrasound, future studies could assess newer or domain-specific CNN variants (e.g. attention-augmented CNNs, medically pretrained transformers) that may better capture low-contrast cues or fine-grained textures than the classic ImageNet trio investigated here.
  - Ensemble learning—combining, for example, a ResNet50 specialised for mammography with a DenseNet121 tuned for ultrasound—could leverage complementary strengths and boost robustness beyond any single architecture.
- **Refining Training Strategies and Addressing Limitations**
  - The initially-unfrozen schedule out-performed progressive fine-tuning for mammography and histopathology **<empty citation>** largely avoiding catastrophic forgetting. A promising extension is to *adaptively* unfreeze layers in response to validation loss, thereby blending the stability of warm-up epochs with the flexibility of end-to-end updates.
  - DenseNet121 and VGG16 exhibited marked degradation under progressive fine-tuning **<empty citation>** discriminative learning rates, stronger regularisation, or longer warm-up phases could mitigate this effect while still permitting modality-specific adaptation.
- **Enhancing Data Diversity and Augmentation**
  - Expanding the training corpus with images drawn from additional demographics, scanner vendors, and clinical sites will improve generalisability and promote equitable performance—key goals for ethical AI deployment.
  - Modality-tailored augmentation (e.g. synthetic variations in breast density for mammography or realistic speckle injection for ultrasound) should be explored to harden models against real-world acquisition variability.
- **Expanding Clinical Utility and Interpretability**

  - Extending the current portal beyond binary classification to include lesion localisation or full segmentation would provide richer, more actionable diagnostics.
  - Integrating explainable-AI artefacts—Grad-CAM heat maps, saliency overlays—within the web interface would help clinicians understand *why* a model reached a decision, enhancing trust and adoption.
  - Longitudinal analysis modules that track lesion evolution across time-series imaging could unlock prognostic insights, echoing recent work on five-year breast-cancer risk prediction.

- **Operationalisation and Scalability**
  - Transitioning the proof-of-concept portal to a production environment entails robust authentication, comprehensive error handling, and seamless PACS/EHR integration while preserving the stateless micro-service architecture introduced in Section **??**.
  - Ultra-low-latency inference for high-throughput clinics could be achieved by leveraging hardware accelerators (AI inference chips, embedded edge devices) and optimised model formats (e.g. TensorRT, ONNX-Runtime).

Bibliography

[1] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, pp. 436–444, 2015. [Online]. Available: https://doi.org/10.1038/nature14539.

[2] D. Rumelhart, G. Hinton, and R. Williams, "Learning representations by back-propagating errors," *Nature*, vol. 323, pp. 533–536, 1986. [Online]. Available: https://doi.org/10.1038/323533a0.

[3] L. G, K. T, B. BE, *et al.*, "A survey on deep learning in medical image analysis," *Medical Image Analysis*, vol. 42, pp. 60–88, 2017. [Online]. Available: https://doi:10.1016/j.media.2017.07.005.

[4] Y. Celik, M. Talo, O. Yildirim, M. Karabatak, and U. R. Acharya, "Automated invasive ductal carcinoma detection based using deep transfer learning with whole-slide images," *Pattern Recognition Letters*, vol. 133, pp. 232–239, 2020, issn: 0167-8655. doi: https://doi.org/10.1016/j.patrec.2020.03.011. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167865520300891.

[5] A. Esteva, B. Kuprel, and R. e. a. Novoa, "Dermatologist-level classification of skin cancer with deep neural networks," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 24, no. 5, pp. 399–418, 1976.

[6] P. Rajpurkar, J. Irvin, K. Zhu, *et al.*, "Chexnet: Radiologist-level pneumonia detection on chest x-rays with deep learning," 2017. arXiv: 1711.05225 [cs.CV]. [Online]. Available: https://arxiv.org/abs/1711.05225.

[7] L. Shen, L. R. Margolies, J. H. Rothstein, E. Fluder, R. McBride, and W. Sieh, "Deep learning to improve breast cancer detection on screening mammography," *Scientific Reports*, vol. 9, no. 1, 2019. doi: 10.1038/s41598-019-48995-4. [Online]. Available: https://doi.org/10.1038/s41598-019-48995-4.

[8] A. Yala, C. Lehman, T. Schuster, T. Portnoi, and R. Barzilay, "A deep learning mammography-based model for improved breast cancer risk prediction," *Radiology*, vol. 292, no. 1, pp. 60–66, 2019. doi: 10.1148/radiol.2019182716. [Online]. Available: https://doi.org/10.1148/radiol.2019182716.

[9] W. McCulloch and W. Pitts, "A logical calculus of the ideas immanent in nervous activity," *Bulletin of Mathematical Biophysics*, vol. 5, pp. 115–133, 1943. [Online]. Available: https://link.springer.com/article/10.1007/BF02478259.

[10] V. Nair and G. Hinton, "Rectified linear units improve restricted boltzmann machines vinod nair," *Proceedings of ICML*, vol. 27, pp. 807–814, Jun. 2010.

[11] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998. doi: 10.1109/5.726791.

[12] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Commun. ACM*, vol. 60, no. 6, pp. 84–90, May 2017, issn: 0001-0782. doi: 10.1145/3065386. [Online]. Available: https://doi.org/10.1145/3065386.

[13] F. Bray, M. Laversanne, H. Sung, *et al.*, "Global cancer statistics 2022: Globocan estimates of incidence and mortality worldwide for 36 cancers in 185 countries," *CA: A Cancer Journal for Clinicians*, vol. 74, no. 3, pp. 229–263, 2024. doi: https://doi.org/10.3322/caac.21834. eprint: https://acsjournals.onlinelibrary.wiley.com/doi/pdf/10.3322/caac.21834. [Online]. Available: https://acsjournals.onlinelibrary.wiley.com/doi/abs/10.3322/caac.21834.

[14] S. Iranmakani, T. Mortezazadeh, F. Sajadian, and et al., "A review of various modalities in breast imaging: Technical aspects and clinical outcomes.," *Egypt J Radiol Nucl Med*, vol. 51, p. 57, 2020. doi: https://doi.org/10.1186/s43055-020-00175-5.

[15] Y. Jiménez-Gaona, M. J. Rodríguez-Álvarez, and V. Lakshminarayanan, "Deep learning based computer-aided systems for breast cancer imaging : A critical review," 2020. arXiv: 2010.00961 [eess.IV]. [Online]. Available: https://arxiv.org/abs/2010.00961.

[16] L. Luo, X. Wang, Y. Lin, *et al.*, "Deep learning in breast cancer imaging: A decade of progress and future directions," 2024. arXiv: 2304.06662 [eess.IV]. [Online]. Available: https://arxiv.org/abs/2304.06662.

[17] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," 2015. arXiv: 1409.1556 [cs.CV]. [Online]. Available: https://arxiv.org/abs/1409.1556.

[18] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," 2015. arXiv: 1512.03385 [cs.CV]. [Online]. Available: https://arxiv.org/abs/1512.03385.

[19] G. Huang, Z. Liu, L. van der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," 2018. arXiv: 1608.06993 [cs.CV]. [Online]. Available: https://arxiv.org/abs/1608.06993.

[20] S. e. a. Naik, "Addressing global gaps in mammography screening for improved breast cancer detection: A review of the literature," *Cureus*, 2024. doi: 10.7759/cureus.66198. [Online]. Available: https://pubmed.ncbi.nlm.nih.gov/39233973/.

[21] N. R. e. a. Payne, "Breast density effect on the sensitivity of digital screening mammography in a uk cohort," *European radiology*, pp. 177–187, 2025. doi: 10.1007/s00330-024-10951-w. [Online]. Available: https://pubmed.ncbi.nlm.nih.gov/39017933/.

[22] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," pp. 248–255, 2009. doi: 10.1109/CVPR.2009.5206848.

[23] M. Heath, K. Bowyer, D. Kopans, R. Moore, and W. P. Kegelmeyer, "The digital database for screening mammography," pp. 212–218, 2009. [Online]. Available: http://www.eng.usf.edu/cvprg/mammography/database.html.

[24] M. Heath, K. Bowyer, D. Kopans, R. M. W. Philip Kegelmeyer, K. Chang, and S. MunishKumaran, "Current status of the digital database for screening mammography," pp. 248–255, 1998. [Online]. Available: http://www.eng.usf.edu/cvprg/mammography/database.html.

[25] "Ddsm mammography," [Online]. Available: https://www.kaggle.com/datasets/skooch/ddsm-mammography.

[26] "Ultrasound breast images for breast cancer," [Online]. Available: https://www.kaggle.com/datasets/paultimothymooney/breast-histopathology-images.

[27] "Breast histopathology images," [Online]. Available: https://www.kaggle.com/datasets/paultimothymooney/breast-histopathology-images.

[28] J. Yosinski, J. Clune, Y. Bengio, and H. Lipson, "How transferable are features in deep neural networks?," 2014. arXiv: 1411.1792 [cs.LG]. [Online]. Available: https://arxiv.org/abs/1411.1792.

[29] M. Raghu, C. Zhang, J. Kleinberg, and S. Bengio, *Transfusion: Understanding transfer learning for medical imaging*, 2019. arXiv: 1902.07208 [cs.CV]. [Online]. Available: https://arxiv.org/abs/1902.07208.

[30] "Transfer learning: Why we freeze and unfreeze model layers," [Online]. Available: https://medium.com/data-science-collective/transfer-learning-why-we-freeze-and-unfreeze-model-layers-0e0b8f9837ec.

[31] S. Kornblith, J. Shlens, and Q. V. Le, *Do better imagenet models transfer better?* 2019. arXiv: 1805.08974 [cs.CV]. [Online]. Available: https://arxiv.org/abs/1805.08974.

[32] Y. Wang, D. Sun, K. Chen, F. Lai, and M. Chowdhury, *Egeria: Efficient dnn training with knowledge-guided layer freezing*, 2023. arXiv: 2201.06227 [cs.LG]. [Online]. Available: https://arxiv.org/abs/2201.06227.

[33] [Online]. Available: https://www.python.org/downloads/release/python-31011/.

[34] [Online]. Available: https://github.com/tensorflow/tensorflow/blob/r2.15/RELEASE.md.

[35] [Online]. Available: https://numpy.org/doc/stable/release/1.26.0-notes.html.

[36] [Online]. Available: https://pandas.pydata.org/docs/whatsnew/v2.2.3.html.

[37] [Online]. Available: https://scikit-learn.org/stable/whats_new/v1.4.html.

[38] [Online]. Available: https://matplotlib.org/stable/users/prev_whats_new/whats_new_3.9.0.html.

[39] [Online]. Available: https://flask.palletsprojects.com/en/stable/changes/.

[40] [Online]. Available: https://docs.opencv.org/4.10.0/.

[41] [Online]. Available: https://seaborn.pydata.org/whatsnew/v0.13.0.html.