# Project 3

## FYS-STK4155

Krithika Gunesegaran, Oskar Idland, Erik Røset & Arangan Subramaniam
*University of Oslo, Department of Physics*
(Dated: December 13, 2024)

Machine learning technologies have had a profound impact in the field of diagnostic imaging and disease detection.[1] Convolutional Neural Networks (CNNs) have demonstrated remarkable success in medical image analysis, but deploying these models effectively requires significant computational resources and large training datasets, presenting a critical challenge for practical medical applications. Here we show through evaluation of three neural architectures (MobileNet, ResNet101 and a architecture of our design) on breast cancer histopathological images that model complexity must be carefully balanced against dataset size. While ResNet101 significantly outperforms other models with higher F1-scores (0.87), simpler architectures trained from scratch showed surprisingly robust performance, suggesting that architectural complexity should be carefully balanced against dataset size in medical applications. This highlights the importance of considering data availability when selecting model architectures for practical medical image classification systems.

https://github.com/Oskar-Idland/FYS-STK4155-Projects

## I. INTRODUCTION

While artificial intelligence has revolutionized many fields, its impact on medical diagnostics has been particularly transformative through automated image analysis systems. The healthcare sector has increasingly adopted Convolutional Neural Networks (CNNs) for critical tasks like tumor detection and disease classification, where both accuracy and reliability are paramount.[2] However, implementing these systems in clinical settings presents unique challenges - traditional neural networks often require substantial computational resources and extensive training data, making them difficult to deploy in resource-limited environments[3]. This has led to the development of various architectural approaches, from lightweight models trained from scratch to deep networks that leverage pre-trained weights from large datasets, each offering different tradeoffs between performance and resource requirements.

To address this, more efficient architectures like MobileNet and ResNet have been created. MobileNet is designed to be lightweight and efficient, making it ideal for tasks like medical image classification. ResNet, on the other hand, uses residual connections to help train deep networks, which solves the degradation problem and improves performance in complex image recognition tasks [4]. Together, these architectures offer a range of solutions for efficient and effective deep learning in medical imaging applications.

This study aims to evaluate and compare the performance of various neural network architectures on a breast cancer histopathological dataset, focusing on three specific models: a custom CNN, MobileNet, and ResNet101. These models are selected based on their differing complexities, training requirements, and computational efficiencies. In this analysis, we will compare how well these models handle a small and imbalanced dataset, aiming to understand how different architectures affect classification accuracy and performance in medical applications.

This report is organized as follows: First, we introduce the theoretical concepts behind neural networks, CNNs, MobileNet, and ResNet, along with the relevant evaluation metrics. Next, we discuss the methods and implementation, including data preprocessing and model architectures. We then present the results, comparing the performance of the models. Finally, we conclude by summarizing our findings and identifying the most suitable model for handling complex data.

## II. THEORY

Neural networks form the foundation of the architectures discussed in this paper. For a comprehensive treatment of neural network fundamentals, including their structure, training process, and mathematical foundations, we refer readers to our previous work.[5] Building upon this foundation, we focus here on the specialized architectures and techniques relevant to our current analysis of medical image classification.

### A. Convolution Neural Network

There are various types of neural networks designed for specific tasks, and among them, Convolution Neural Networks (CNN) are one of the most widely used and effective. CNN were originally created for tasks like recognizing handwritten characters. Today, CNNs are more central to advancements in artificial intelligence, particularly in computer vision tasks and applications, where they often leverage image processing techniques [6]. CNNs are similar to traditional Neural Networks in that they consist of neurons with adjustable weights that learn patterns from data.

CNNs are great at using the natural structure of visual data. Unlike other types of data, images have unique features like edges, textures, and shapes. CNNs are specif-

ically designed to recognize and process these features efficiently, requiring less parameters than the traditional neural networks [6]. This efficiency makes CNNs faster and easier to use on large tasks, and very effective for things like object detection and face recognition. They are specifically built for image data, with a structure that aligns with the way images are organized. Unlike regular neural networks, CNN have neurons arranged in 3 dimensions: width, height, and depth [7].

To understand why CNNs are so effective, it's important to first examine how neural networks process data. Neural networks commonly use an affine transformation, a method that works for any type of data that can be flattened into a single list of numbers [8]. However, for structured data like images, this approach has a significant drawback because it ignores the natural structure of the data. Images are multi-dimensional grids of pixels where the spatial arrangement, such as width, height and depth is crucial [7]. Flattening these grids into a single vector removes this valuable information. This is where convolutions layers is essential.

### 1. *Convolutional layers*

Convolutional layers are a key part of Convolutional Neural Networks (CNNs), designed to handle structured data like images while keeping their spatial relationships [7]. Convolutional layers use the convolution operation to turn raw data into feature maps that highlight important patterns like edges, textures, and shapes. This helps CNNs learn and understand complex features, making them essential for tasks like image classification and object detection. The convolution operation is mathematically described as:

$$y(t) = \int x(a)w(t-a)\,da \tag{1}$$

Where $x(a)$ is the input data (e.g., an image), $w(t-a)$ is the filter or kernel that moves across the input, and $y(t)$ is the output of the convolution, capturing the features extracted from the input. For digital data, this integral becomes a discrete summation:

$$y(t) = \sum_{a=-\infty}^{\infty} x(a)w(t-a) \tag{2}$$

In practice, the filter slides over the input data spatially, computing a dot product at each position [7]. The result is a feature map that highlights the presence of specific patterns associated with the filter. The spatial dimensions of the output feature map are determined as:

$$W_{\text{output}} = \frac{W_{\text{input}} - F + 2P}{S} + 1 \tag{3}$$

$$H_{\text{output}} = \frac{H_{\text{input}} - F + 2P}{S} + 1 \tag{4}$$

$$D_{\text{output}} = K \tag{5}$$

Here, $W_{\text{input}}$ and $H_{\text{input}}$ are the width and height of the input, $F$ is the filter size, $P$ is the padding (extra pixels added around the input to preserve edge information), and $S$ is the stride (the step size of the filter). The depth of the output volume corresponds to the number of filters $K$, with each filter learning to detect a specific feature.

Convolutional layers achieve computational efficiency through parameter sharing, where all neurons within a filter share the same weights and biases [7]. This significantly reduces the number of learnable parameters compared to fully connected layers, enabling CNNs to process large datasets efficiently.

Key parameters such as filter size $F$, padding $P$, stride $S$, and the number of filters $K$, optimize the layer's functionality [7]. Smaller filters capture fine details, padding keeps edge information, and strides controls how much the filter moves. The number of filters sets the output depth, with each filter learning different features. These parameters, along with the mathematical formula for output dimensions, makes convolutional layers essential to modern computer vision.

### 2. *Kernel*

Convolutional kernels, also called filters, are critical components of convolutional neural networks (CNNs), performing feature extraction through sliding-window operations over input feature maps [9]. Each kernel is defined by its dimensions and weights, which are refined during training to detect specific features in the input data. The application of kernels produces feature maps by combining the input image with the learned filters. This process reduces the number of trainable parameters, enhancing the efficiency of training and inference. The spatial dimensions of the output feature map can be calculated using the formula $W_{\text{out}} = W - F + 2P/S + 1$, where $W$ is the input width, $F$ is the kernel size, $P$ is the padding, and $S$ is the stride. This shared-weight scheme inherent in kernel operations not only accelerates network computation but also improves generalization, making CNNs highly effective for computer vision tasks [10].

### 3. *Pooling layers*

In CNN architecture, there is no direct communication between layers that allows information to flow backward [11]. As a result, each layer processes the data independently. However, it is crucial for the pooling layer to

pass relevant information to the next layer. Pooling plays an important role by reducing the spatial dimensions of the feature maps, making them more manageable while retaining key information. There are different types of pooling methods, such as average pooling and max pooling, each with its own characteristics and applications.

**Max pooling**

Max pooling is a commonly used operation in CNN for feature extraction. The main purpose of max pooling is to reduce the spatial dimensions of feature maps, which helps make the network more computationally efficient while preserving important information.

In max pooling, the input image is divided into small regions or windows (e.g., a 2x2 or 3x3 window). From each window, max pooling selects the maximum value and uses it as the representative value for that region. This allows the network to focus on the most prominent features in the image, such as edges, textures, and shapes, while discarding less important details. As a result, max pooling helps preserve important local features. Max pooling is particularly effective at preserving sharp, local features like edges and corners, making it highly useful in image processing and computer vision tasks [11].

However, this operation results in some information loss, as it only keeps the maximum value from each window and discards the rest. While this makes the data more compact, it can also lead to a reduction in accuracy by losing finer details of the image. This is mathematically expressed as:

$$y = \max(x_1, x_2, \ldots, x_n) \tag{6}$$

Where $x_1, x_2, \ldots, x_n$ represent the values in the pooling region, and $y$ is the output value for that region.

### B. MobileNet

MobileNet is a lightweight and efficient deep neural network architecture designed specifically for mobile and embedded vision tasks. It reduces computational complexity and memory usage by employing depthwise separable convolutions instead of traditional convolutions. This involves splitting the standard convolution process into two distinct steps: a depthwise convolution,which applies a single filter to each input channel, and a pointwise convolution (1×1 convolution), which combines the outputs to form new features [3]. This factorization drastically reduces the computational cost and model size, making MobileNet an ideal choice for resource-constrained devices.

The depthwise convolution operation in MobileNets can be mathematically represented as follows:

$$\hat{G}_{k,l,m} = \sum_{i,j} \hat{K}_{i,j,m} \cdot F_{k+i-1,l+j-1,m} \tag{7}$$

Here, $\hat{G}_{k,l,m}$ is the output value at spatial location $(k, l)$ in the $m$-th output channel. The filter $\hat{K}_{i,j,m}$ is applied to the corresponding region of the input feature map $F$ with the position $(k+i-1, l+j-1)$, where $i$ and $j$ represent the spatial coordinates within the kernel. The summation $(\sum_{i,j})$ aggregates the product of each filter weight and its corresponding input value across the kernel's spatial dimensions, producing a filtered output for each input channel. depthwise convolution thus filters each input channel independently, preserving spatial relationships within each channel but not combining information across channels, which is later handled by pointwise convolution in the depthwise separable convolution process [3].

Pointwise convolution refers to a $1 \times 1$ convolution that combines the outputs of the depthwise convolution across channels to create new feature representations . It operates after depthwise convolution, where each input channel is filtered independently, and then a $1 \times 1$ convolution mixes the results across all channels [3]. This process reduces computational cost significantly while maintaining the ability to learn complex features by mixing the depthwise outputs efficiently [4].

### C. Residual Networks (ResNet)

Deep neural networks faces two major challenges: the degradation problem and the vanishing/exploding gradients problem. As networks get deeper, their performance can reach a limit, and adding more layers may actually increase training errors, a phenomenon known as the degradation problem. Additionally, during backpropagation, gradients may become extremely small (vanishing) or excessively large (exploding), which hampers the effective training of very deep networks [4].

Residual Networks (ResNet) address these challenges by introducing a residual learning framework. Instead of directly approximating the desired mapping $H(x)$, ResNet reformulates the task to learn a residual function $F(x) = H(x) - x$, where $x$ is the input. The final output is computed as $H(x) = F(x) + x$, which simplifies optimization, especially when the optimal function is close to an identity mapping [4]. This reformulation alleviates the degradation problem by preconditioning the learning process.

At the core of ResNet is the Residual block, which incorporates a skip connection (shortcut) that directly adds the input $x$ to the output of the residual function, defined as $y = F(x, \{W_i\}) + x$. These skip connections facilitate gradient flow during backpropagation, addressing the vanishing gradient problem, and require no additional parameters or computational overhead. If the input and output dimensions differ, a linear projection $W_s x$ can be used ($y = F(x, \{W_i\}) + W_s x$), though identity mappings are often sufficient. Residual blocks can be applied to both fully connected and convolutional layers, enabling the construction of deeper and more scalable networks.

By retaining information from earlier layers and simplifying the learning of deeper representations, ResNet has become a cornerstone of modern deep learning architectures [4].

ResNet architectures are composed of multiple stacked residual blocks, making it possible to build very deep networks. Popular variants include:

- ResNet-50: A 50-layer network [4].
- ResNet-101: A deeper network with 101 layers [4].
- ResNet-152: An even deeper variant with 152 layers [4].

These architectures typically begin with an initial convolutional layer, followed by residual blocks grouped into stages, and conclude with a fully connected layer for classification tasks.

Before the final fully connected layer, ResNet employs Global Average Pooling (GAP) to condense each feature map into a single value [4]. GAP provides a compact representation of the feature maps, improving computational efficiency and reducing the risk of overfitting [4].

ResNet achieves state-of-the-art performance on tasks like image classification, particularly in the ImageNet Large Scale Visual Recognition Challenge (ILSVRC). For instance, ResNet-152 achieved a top-5 error rate of 3.57% on ImageNet. By enabling the training of networks with hundreds or even thousands of layers, ResNet solved critical challenges in deep learning and set a new standard for neural network design [4].

### D. Transfer Learning

Transfer learning enables neural networks to leverage knowledge gained from one task to improve performance on another. This approach is particularly valuable when working with limited datasets, as it allows models to build upon features already learned from much larger datasets rather than learning everything from scratch.

In deep neural networks like ResNet101, the early layers typically learn basic visual features such as edges, textures, and simple shapes, while deeper layers learn more task-specific features. These basic visual features are often useful across different types of image recognition tasks, even when the domains are quite different (such as natural images versus medical images). This makes it possible to use models pre-trained on large datasets like ImageNet as a starting point for more specific tasks.

When applying transfer learning, the pre-trained model's weights serve as an intelligent initialization point instead of random initialization. The model can then be fine-tuned on the target dataset, allowing it to adapt its learned features to the specific task while retaining the beneficial general features it has already learned. This approach is especially useful in medical imaging applications where labeled data is often limited, as it reduces the amount of data needed to achieve good performance.

The effectiveness of transfer learning helps explain why a deep, pre-trained model like ResNet101 might perform differently from simpler models trained from scratch when working with limited datasets. While simpler models must learn all their features from the available data, pre-trained models begin with a rich set of relevant features that need only be refined for the specific task.

### E. Evaluation Metrics

#### 1. *Confusion Matrix*

The metrics are derived from the confusion matrix, which provides a detailed breakdown of model predictions:

- True Positives (TP): Correctly predicted positive instances.
- True Negatives (TN): Correctly predicted negative instances.
- False Positives (FP): Negative instances incorrectly predicted as positive.
- False Negatives (FN): Positive instances incorrectly predicted as negative.

The confusion matrix serves as the foundation for most classification metrics, offering insights into both model strengths and weaknesses [12, 13].

#### 2. *Precision*

Precision measures the proportion of true positive predictions among all instances predicted as positive. This metric is particularly important in scenarios where false positives carry significant costs, such as in fraud detection or medical testing. A high precision indicates that the model has effectively minimized false positive predictions [12]. Precision is mathematically expressed as:

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} \tag{8}$$

Precision is a valuable metric for evaluating classification models, particularly in cases involving imbalanced datasets. It measures the model's correctness in identifying the target class, making it effective for scenarios where false positives carry significant costs [14]. For example, in medical diagnostics, precision is crucial for confirming diagnoses like cancer, ensuring minimal unnecessary treatments due to false positives [15]. Precision works well when correctly identifying the negative class is less critical and is most suited for binary classification tasks [16].

However, precision has notable limitations. It does not consider false negatives, meaning it overlooks cases where

the target event is missed, which can be critical in many applications [14, 17]. This lack of consideration for false negatives means that precision alone does not provide a complete picture of model performance . Additionally, an excessive focus on precision can lead to overly conservative models that avoid predicting the positive class to maintain a high precision score, potentially missing many true positives . For these reasons, precision should be used alongside other metrics, such as recall, to gain a holistic understanding of the model's performance [17].

### 3. Recall

Recall quantifies the ability of the model to identify actual positive cases. It is critical in applications where missing positive cases (false negatives) have severe implications, such as in medical diagnoses or safety-critical systems [12].

This is given as:

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} \tag{9}$$

Recall is a critical metric for evaluating model performance, particularly in applications involving imbalanced datasets. It highlights how well a model identifies the minority class, helping us understand its performance in such scenarios. Recall also provides insights into potential biases in the data, showcasing how much the data or model leans toward a specific class [16]. Additionally, it evaluates the efficiency of the model in capturing true positives, making it highly relevant in tasks where missing positive cases can have significant consequences, such as fraud detection or diagnostic medical tests [14, 15].

However, recall has limitations. It only considers true positives and false negatives, ignoring true negatives, which means it doesn't provide a complete picture of model performance across all aspects of the confusion matrix . Furthermore, recall focuses exclusively on the positive class, making it less suitable in situations where correctly identifying negative classes is also important . This metric is best suited for binary classification tasks, where the focus is on maximizing true positive rates [16].

Recall is especially valuable in cases like fraud detection, where identifying as many fraudulent cases as possible is critical, even at the cost of some false positives [15]. Similarly, in medical diagnostics, recall ensures that most cases of a condition, such as a disease, are flagged, minimizing the risk of missing affected individuals [14] . It is particularly effective in situations where identifying the negative class is not a priority [16] .

### 4. Accuracy

Accuracy measures the overall proportion of correctly classified instances among all predictions. While it is easy to compute and interpret, accuracy can be misleading in imbalanced datasets where one class dominates, making additional metrics like the F1-score necessary [12, 13]. This is expressed as:

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{Total Instances}} \tag{10}$$

Accuracy is a widely used metric for evaluating classification models due to its simplicity and ease of use. It is straightforward to understand, making it an intuitive choice for many practitioners. Additionally, accuracy works well for balanced datasets, providing a single value that allows for easy comparison of model performance [16].

However, accuracy has several notable limitations. It can be misleading when applied to imbalanced datasets . For instance, in cases where 95% of the samples belong to one class, a model predicting only the majority class will achieve high accuracy but perform poorly in identifying the minority class, undermining its overall utility [15]. Furthermore, accuracy does not offer insights into the specific types of errors made by the model, such as false positives or false negatives [16]. It also fails to consider the probability scores of predictions, limiting its interpretability compared to metrics like the confusion matrix [16]. While accuracy is suitable for balanced datasets [18, 19], it may not be the best choice for evaluating models in scenarios involving imbalanced data distributions [19].

### 5. F1-Score

The F1-score provides a harmonic mean of Precision and Recall, offering a balanced evaluation metric that is particularly useful in datasets with imbalanced class distributions. It penalizes extreme differences between Precision and Recall, ensuring a more holistic assessment of model performance [12]. This can be expressed as:

$$\text{F1-Score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \tag{11}$$

The F1-Score is a widely used evaluation metric that combines precision and recall into a single value, making it suitable for comparing models [20]. By using the harmonic mean, the F1-score ensures that the result reflects the lower of the two values, effectively emphasizing weaknesses when either precision or recall is low [16]. This makes the F1-Score particularly effective in imbalanced datasets where the balance between these metrics is crucial for a holistic understanding of performance [15].

However, the F1-Score has its limitations. It provides no insight into the distribution of errors, which can be critical for applications requiring detailed error analysis, such as medical diagnostics or fraud detection. Additionally, it assumes that precision and recall have equal importance, which may not align with real-world scenarios

where the costs or significance of false positives and false negatives differ . The F1-Score is also a generic metric, meaning it does not account for specific data patterns or unique problem characteristics, potentially leading to an incomplete evaluation of the model's performance. Furthermore, it is primarily designed for binary classification and may require adaptation for multi-class problems, such as micro or macro averaging, to be meaningful in those contexts [20].

The F1-Score is particularly effective for applications with imbalanced datasets, where precision and recall alone might fail to reflect the model's performance accurately. For instance, in detecting rare events like fraud or disease, the F1-Score can provide a more balanced and reliable measure of model effectiveness [15].

## III. METHODS & IMPLEMENTATION

### A. Data Preprocessing and Model Architecture

The dataset consisted of breast cancer histopathological images classified into three categories: normal, malignant and benign. Each malignant and benign image included an associated mask file denoting regions of interest, which were excluded from this analysis. All images were preprocessed to a standardized size of 224 × 224 pixels to maintain consistency with standard deep learning architectures while preserving sufficient detail for medical diagnosis.

The dataset was partitioned into training (563 images), validation (100 images), and test (117 images) sets, representing approximately 72%, 12% and 15% of the data respectively. Stratification was maintained across splits to preserve class distribution, particularly important given the significant class imbalance (benign: 437, malignant: 210, normal: 133). Data augmentation techniques were employed to address this imbalance, specifically targeting the minority classes (malignant and normal). The augmentation pipeline included random horizontal flips with a high probability (p=0.9) to maximize the diversity of cell orientations, constrained rotation (±15°) to maintain anatomical plausibility, and color jittering to enhance robustness against staining variations common in histopathological samples.

Three distinct deep learning architectures were implemented and compared:

- A custom simple CNN was designed with specific consideration for the small dataset size. The architecture comprised two convolution layers ($3 \rightarrow 16 \rightarrow 32$ channels) with $3 \times 3$ kernels, chosen to capture local cellular features while maintaining computational efficiency. ReLU activation was selected for its non-linearity and reduced likelihood of vanishing gradients. Max pooling operations follow each convolution layer to achieve spatial dimensionality reduction while preserving important features. The resulting $56 \times 56 \times 32$ feature maps are flattened into a 100,352-dimensional vector, followed by fully connected layers reducing to 512 and 128 neurons respectively. This progressive reduction in dimensionality ($100,352 \rightarrow 512 \rightarrow 128 \rightarrow 3$) creates an information bottleneck that forces the network to learn increasingly abstract representations of the input data, with the final layer outputting probabilities for our three classes.

- A MobileNet architecture[21], selected for its efficient depth-wise separable convolutions which significantly reduce computational complexity while maintaining performance. This architecture has demonstrated success in medical image classification tasks where computational resources may be limited in clinical settings.[2]

- A pre-trained ResNet101 model[22], chosen for its deep residual learning framework that effectively addresses the degradation problem in deep networks and general good performance in medical classification tasks[23]. The model was pre-trained on ImageNet, allowing it to leverage general feature detection capabilities, while its final layer was modified to accommodate our three-class classification task.

### B. Training and Evaluation

The models were trained using the Adam optimizer, selected for its adaptive learning rate capabilities and robust performance across different neural architectures. Learning rates were empirically determined: 0.001 for both Simple CNN and MobileNet, and a lower rate of 0.00005 for ResNet101 to prevent catastrophic forgetting of pre-trained features. A step learning rate scheduler (step size=7, $\gamma$=0.1) was implemented to facilitate convergence to better optima. Training utilized cross-entropy loss, appropriate for our multi-class classification task, and incorporated early stopping (patience=2, minimum epochs=5) to prevent overfitting while ensuring sufficient learning time.

Models were evaluated using a comprehensive set of metrics: accuracy, precision, recall, and F1-score, with particular attention to per-class performance given the imbalanced nature of our dataset. Confusion matrices were generated to provide detailed insight into class-wise performance and misclassification patterns.

### C. Implementation Details

The implementation used the PyTorch framework and was executed on GPU hardware. Training was conducted with a batch size of 8, chosen to balance between computational efficiency and stable gradient updates. Data normalization used ImageNet statistics (mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225]) to ensure compatibility with pre-trained models and standardize feature scales. All experiments were conducted with a fixed random seed for reproducibility, with final evaluation performed on the held-out test set to ensure unbiased performance assessment.

### D. Tools

Our code is written in Python (3.10) [24], we used scikit-learn [25] and PyTorch[26] to test against our models. To vectorize our code we used `numpy` [27], and for visualization we used `matplotlib.pyplot` [28]. All python packages and their versions can be found in our `requirements.txt`. Code development was done in Visual Studio Code [29] with additional assistance of GitHub Copilot [30]. We used `git` [31] for version control, and `GitHub` [32] for remote storage of our code.

## IV. RESULTS & DISCUSSION

We present a comprehensive analysis of three neural network architectures' performance on the breast cancer histopathological dataset, examining their effectiveness in handling a relatively small medical imaging dataset (780 images). Our analysis spans model accuracy, class-wise performance, and training dynamics to understand how architectural complexity and transfer learning influence classification performance under data constraints.

### A. Model Performance Overview

We present our analysis through confusion matrices (fig. 1) and classification metrics (table I), which reveals distinct patterns in how each architecture handles the classification task. ResNet101 demonstrates superior and balanced performance across all classes, while both the Simple CNN and MobileNet show characteristic behaviors reflecting their architectural differences and the challenges of limited training data.

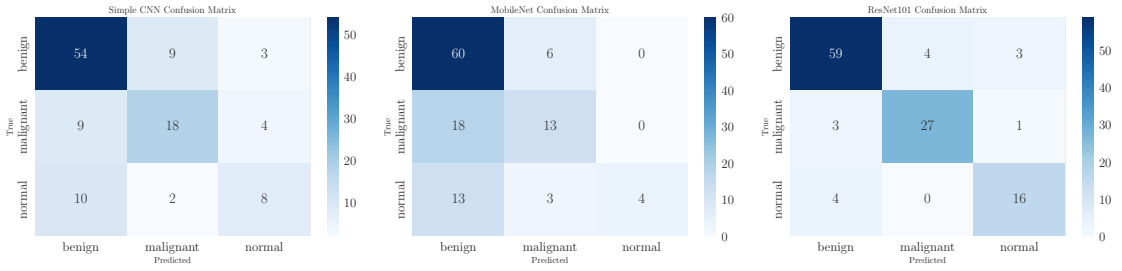


FIG. 1: Confusion matrices for the three neural network architectures (Simple CNN, MobileNet, and ResNet101) evaluated on the breast cancer histopathological test dataset. Values and color intensity indicate the number of images classified in each category.

The confusion matrices demonstrate distinct classification behaviors across architectures. ResNet101 exhibits balanced prediction distribution across classes, while both the simple CNN and MobileNet display systematic bias toward the majority class (benign), as evidenced by the disproportionate predictions in their respective primary columns.

ResNet101 shows markedly superior behavior compared to both other models, achieving not just higher but more consistent performance across all classes. The confusion matrix demonstrates balanced prediction patterns: correctly identifying 59/66 benign, 27/31 malignant, and 16/20 normal cases. This balanced performance suggests that the pre-trained features, developed on a large diverse dataset, provide a robust foundation that can be effectively fine-tuned even with limited medical imaging data.

The Simple CNN, despite its basic architecture, demonstrates more balanced performance across classes than MobileNet, though with lower overall accuracy. Its confusion matrix reveals consistent prediction patterns, correctly identifying 54 of 66 benign cases (0.82 recall)

while maintaining modest but balanced performance on minority classes (malignant: 18/31, normal: 8/20 correct identifications). This suggests that simpler architectures might be more robust to class imbalance when training data is limited, possibly due to fewer parameters needing optimization and thus less opportunity for overfitting to the majority class.

Of particular note is MobileNet's handling of the normal class, where it shows perfect precision (1.00) but poor recall (0.20). This seemingly contradictory result stems from the model's extreme conservatism in predicting the normal class: out of 117 test cases, it only predicted "normal" 4 times. While all of these predictions were correct (hence the perfect precision), the model failed to identify 16 out of 20 normal cases, instead classifying them primarily as benign (13 cases) or malignant (3 cases). This behavior suggests that despite our data augmentation efforts, MobileNet struggled with the class imbalance in the training data, adopting an overly conservative strategy for the minority class.

| | Simple CNN | MobileNet | ResNet101 |
|---|---|---|---|
| **Precision** | | | |
| Benign | 0.74 | 0.66 | 0.89 |
| Malignant | 0.62 | 0.59 | 0.87 |
| Normal | 0.53 | 1.00 | 0.80 |
| *Weighted Avg.* | 0.67 | 0.70 | 0.87 |
| **Recall** | | | |
| Benign | 0.82 | 0.91 | 0.89 |
| Malignant | 0.58 | 0.42 | 0.87 |
| Normal | 0.40 | 0.20 | 0.80 |
| *Weighted Avg.* | 0.68 | 0.66 | 0.87 |
| **F1-score** | | | |
| Benign | 0.78 | 0.76 | 0.89 |
| Malignant | 0.60 | 0.49 | 0.87 |
| Normal | 0.46 | 0.33 | 0.80 |
| *Weighted Avg.* | 0.68 | 0.62 | 0.87 |

TABLE I: Classification metrics grouped by metric type, comparing performance across models. The weighted averages account for class imbalance in the dataset.

### B. Architectural Performance Analysis

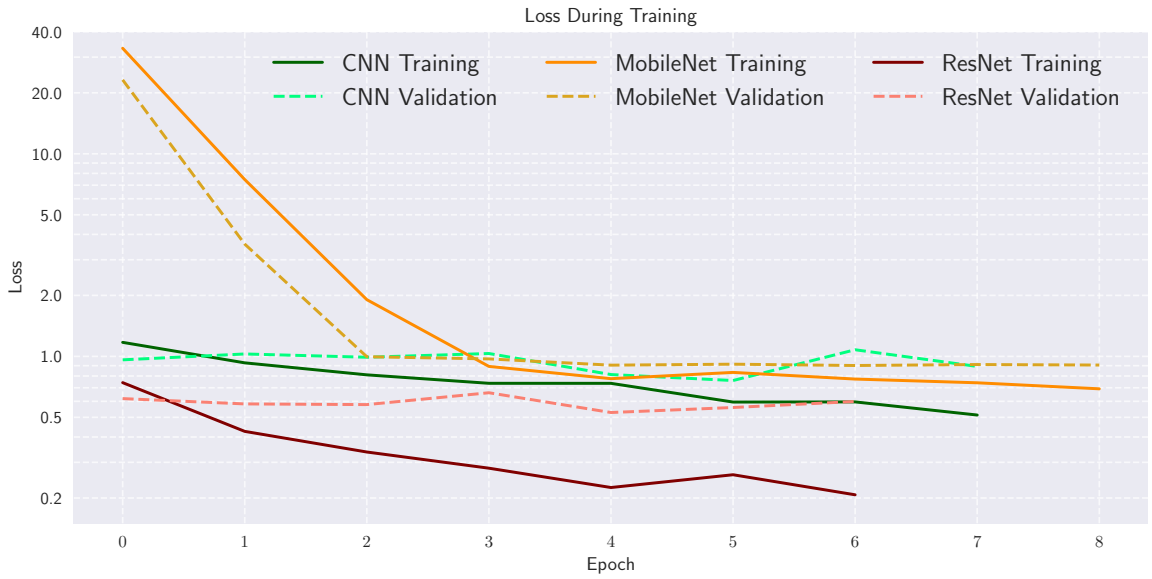


FIG. 2: Loss values during model training plotted on a logarithmic scale. Solid lines represent training loss and dashed lines represent validation loss for each architecture over training epochs.

The training dynamics of each model architecture (fig. 2 and fig. 3) provide additional insight into model convergence and learning efficiency. ResNet101 initiated training with lower loss values (0.74) compared to the Simple CNN (1.17) and MobileNet (33.19), attributable to its pre-trained weights. The convergence trajectories also differ significantly: ResNet101 exhibited steady descent to a final loss of 0.23, while MobileNet required several epochs to stabilize from its initially high loss values. The Simple CNN maintained intermediate loss values throughout training, converging to 0.59.

Accuracy measurements correlate with these observations. ResNet101 achieved initial validation accuracy of 0.74, improving to 0.83 through training. In contrast, the Simple CNN and MobileNet started at 0.56 and 0.33 respectively, with final validation accuracies of

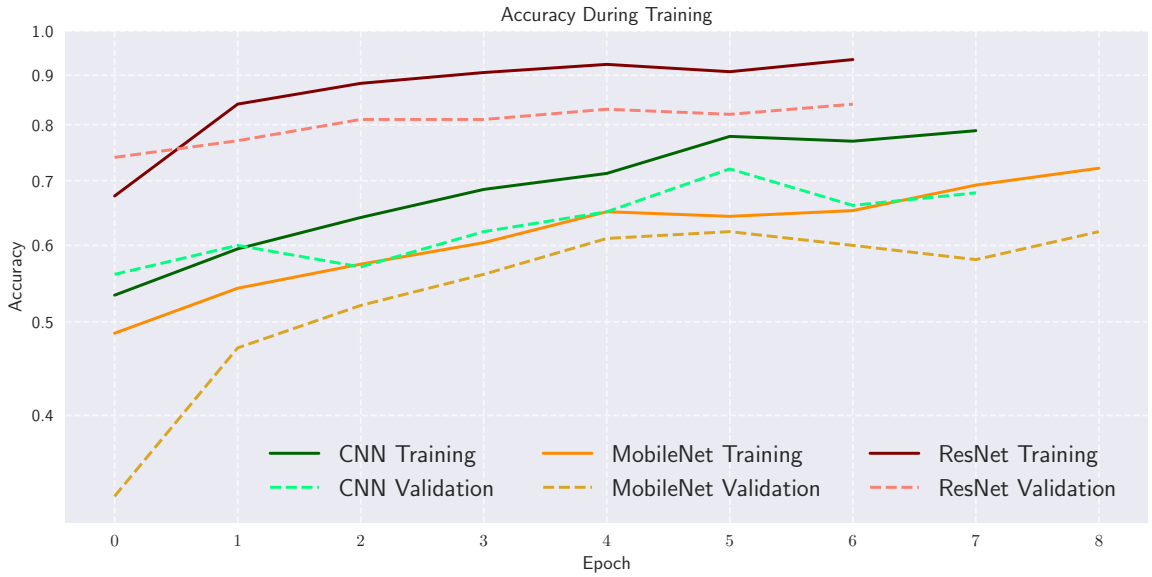


FIG. 3: Model accuracy during training. Solid lines show training accuracy and dashed lines show validation accuracy for each architecture over training epochs.

0.72 and 0.60. Early stopping activated at 7-8 epochs across all architectures, and the proximity between validation and training curves suggests minimal overfitting in all cases. This rapid convergence indicates that performance limitations stem from architectural and data constraints rather than insufficient training duration.

## V. CONCLUSION

These experimental results yield several crucial insights for developing practical medical image classification systems:

- **Transfer Learning Effectiveness:** Pre-trained models can significantly outperform custom architectures on small datasets, with ResNet101 demonstrating nearly 20% higher F1-scores (0.87) than models trained from scratch (CNN: 0.68, MobileNet: 0.62).
- **Architectural Complexity Trade-offs:** The superior performance of the Simple CNN compared to MobileNet contradicts the general assumption that more complex architectures yield better results. This finding suggests that architectural complexity must be carefully balanced against dataset size, particularly in medical imaging applications where large, annotated datasets are often unavailable.
- **Class Imbalance Challenges:** The degraded performance of both Simple CNN and MobileNet on minority classes, despite data augmentation efforts, highlights persistent challenges in handling class imbalance in medical datasets. ResNet101's more balanced performance suggests that transfer learning can help mitigate these issues.
- **Resource Considerations:** While transfer learning from large-scale pre-trained models offers superior performance, the requirements for computational resources and initial training data may limit accessibility. The adequate performance of simpler architectures suggests that targeted, dataset-appropriate model design might offer a more practical solution in resource-constrained settings.

These findings demonstrate that successful medical image classification with limited data requires careful consideration of model complexity, transfer learning opportunities, and computational constraints rather than defaulting to more complex architectures. Future work should investigate methods to improve minority class performance and explore architectural modifications that better utilize limited training data.

For future work, we suggest several methodological enhancements to build upon our current findings. Implementation of cross-validation techniques, particularly k-fold validation, would provide more robust performance estimates and better insight into model stability across different data partitions. Evaluation of additional architectural variants, such as VGGnet or EfficientNet, could offer new perspectives on the relationship between model complexity and performance on small medical datasets. Systematic hyperparameter optimization through techniques like grid search or Bayesian optimization could further improve model performance within the existing architectures. Additionally, we suggest exploring

more sophisticated data augmentation strategies, such as mixup or cutmix, specifically tailored for histopathological images to better address class imbalance and enhance model generalization.

## REFERENCES


[1] G. Litjens, T. Kooi, B. E. Bejnordi, A. A. A. Setio, F. Ciompi, M. Ghafoorian, J. A. Van Der Laak, B. Van Ginneken, and C. I. Sánchez, Medical image analysis **42**, 60 (2017).
[2] P. Sharma, D. R. Nayak, B. K. Balabantaray, M. Tanveer, and R. Nayak, Neural Networks **169**, 637 (2024).
[3] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," (2017), arXiv:1704.04861 [cs.CV].
[4] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," (2015), arXiv:1512.03385 [cs.CV].
[5] E. J. Røset and O. Idland, "Project 2: Fys-stk4155," https://github.com/Oskar-Idland/FYS-STK4155-Projects/tree/main/Project_2 (2024).
[6] AI and Insights, "Convolutional neural networks (cnns) in computer vision," (2023), accessed: 2024-12-12.
[7] C. C. N. N. for Visual Recognition, "Convolutional neural networks (cnns / convnets)," https://cs231n.github.io/convolutional-networks/ (n.d.), accessed: 2024-12-12.
[8] DeepAI, "Affine layer," https://deepai.org/machine-learning-glossary-and-terms/affine-layer (n.d.), accessed: 2024-12-12.
[9] A. Jain, "All about convolutions, kernels & features in cnn," https://medium.com/@abhishekjainindore24/all-about-convolutions-kernels-features-in-cnn-c656616390a1 (2020), accessed: 2024-12-13.
[10] D. Bhatt, C. Patel, H. Talsania, J. Patel, R. Vaghela, S. Pandya, K. Modi, and H. Ghayvat, Electronics **10**, 2470 (2021).
[11] L. Zhao and Z. Zhang, Scientific Reports **14**, 1589 (2024).
[12] M. Hossin and S. M.N, International Journal of Data Mining & Knowledge Management Process **5**, 01 (2015).
[13] H. Dalianis, *Clinical text mining: Secondary use of electronic patient records* (Springer Nature, 2018).
[14] E. AI, "Accuracy, precision, and recall: A guide for monitoring classification models," (n.d.), accessed: 2024-12-12.
[15] A. A. Course, "Evaluation metrics in machine learning," (n.d.), accessed: 2024-12-12.
[16] B. Bose, "Performance metrics for classification models," (n.d.), accessed: 2024-12-12.
[17] P. AI, "Precision and recall: Everything you need to know," (n.d.), accessed: 2024-12-12.
[18] Cohere, "Classification evaluation metrics: Everything you need to know," (n.d.), accessed: 2024-12-12.
[19] B. T. Duong, "Machine learning evaluation metrics," (n.d.), accessed: 2024-12-12.
[20] Serokell, "A guide to the f1 score: Definition, formula, and explanation," (n.d.), accessed: 2024-12-12.
[21] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, CoRR **abs/1704.04861** (2017), 1704.04861.
[22] T. P. Foundation, "Resnet101," https://pytorch.org/vision/main/models/generated/torchvision.models.resnet101.html (Accessed: November 2024).
[23] Y. Wang, Z. Feng, L. Song, X. Liu, and S. Liu, Computational and Mathematical Methods in Medicine **2021**, 2485934 (2021), https://onlinelibrary.wiley.com/doi/pdf/10.1155/2021/2485934.
[24] P. S. Foundation, "Python 3.11.4 documentation," https://docs.python.org/3.11/ (Accessed: October 2024).
[25] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, Journal of Machine Learning Research **12**, 2825 (2011).
[26] T. P. Foundation, "Pytorch," https://pytorch.org/docs/stable/index.html (Accessed: November 2024).
[27] N. Developers, "Numpy v1.22 manual," https://numpy.org/doc/stable/ (Accessed: October 2024).
[28] M. D. Team, "Matplotlib: Visualization with python," https://matplotlib.org/stable/index.html (Accessed: October 2024).
[29] Microsoft, "Visual studio code," https://code.visualstudio.com/docs (Accessed: October 2024).
[30] Microsoft, "Github copilot," https://copilot.github.com/ (Accessed: October 2024).
[31] J. Hamano, "Git," https://git-scm.com/doc (Accessed: October 2024).
[32] Microsoft, "Github," https://docs.github.com/en (Accessed: October 2024).

## Appendix A: Code

Link to our GitHub repository: https://github.com/Oskar-Idland/FYS-STK4155-Projects