Enhancing Computer-Aided Diagnosis for Breast Cancer Detection: An AI Approach Utilizing ResNet50 and Logistic Regression for Mammography Classification

Siem Fenne (VU Student Number: 2654408)

supervisor: prof. dr. M. Massmann
co-reader: dr. Lukas Hoesch

April 1, 2024

Abstract

Breast cancer is still among the most diagnosed non-skin cancer among females, necessitating the importance of accurate detection methodologies. In this study, we proposed a deep learning approach utilizing ResNet50, a state-of-the-art convolutional neural network, for the classification of mammography breast mass lesion images. We conducted a comparative study between the ResNet50 model and a lasso regularized logistic regression, while addressing interpretability of both models. Integrating transfer learning and fine-tuning, we found that the ResNet50 model outperformed the logistic regression model, achieving an accuracy, sensitivity, and specificity of 0.79, 0.73, and 0.84 respectively for the ResNet50 model, against 0.61, 0.54, and 0.65 respectively for the logistic regression. To address the interpretability of the ResNet50 model, we visualized intermediate activations, kernels, and heatmaps of class activation using Gradient-weighted Class Activation Mapping. This allowed for understanding the decision making process of the ResNet50 model, offering potentially useful insights for radiologists. This study shows the potential of deep learning approaches over classic statistical methods in complex image recognition tasks such as breast cancer detection, and paves the way for more accurate and interpretable computer-aided diagnosis systems in the fight against breast cancer.

Keywords: Breast cancer detection, Deep learning, Mammography, Computer-aided diagnosis.

VU VRIJE UNIVERSITEIT AMSTERDAM

Contents

1 Introduction

In the past decades, breast cancer has consistently been ranked the most frequently diagnosed non-skin cancer and the leading cause of death among females, having a mortality rate of 32% (Ferlay et al. 2021). In fact, the American Cancer Society has predicted that in 2025, more than 290,000 new cases will be reported, with 43,780 woman expected not to survive the decease (Siegel et al. 2022). Early detection of the decease is crucial, as five-year relative survival rates can be up to 3-4 times higher when the decease is detected at an early stage versus later stages (National Cancer Institute 2023).

An important tool for the early detection of breast cancer is mammography screening, a method used by radiologists that has been attributing to a significant reduction of the mortality rate. However, one of the limiting factors of mammography is the time-consuming nature of screening, especially with the increasing number of daily screened mammograms (Baccouche et al. 2022). Additionally, screening mammograms is an error-prone method, and not all radiologists achieve the desired accuracy (Elmore et al. 2009). Therefore, effective machine vision-based tools focusing on mammography screening hold high potential by assisting radiologists and help improve the outcome of the decease.

Such tools, often categorized as computer-aided diagnosis (CAD) systems, traditionally consist of a sequence of stages. The first stage includes the detection and localization of suspicious lesions while simultaneously distinguishing the type of lesions, i.e. architectural distortion, calcification, or mass. Then, in the second stage, the system applies a segmentation technique to obtain the region of interest (ROI) surrounding the breast lesion while removing part of the mammogram which is not of interest. And finally, diagnostic information is extracted regarding the lesion's pathology as either malignant or benign using a learning algorithm (Lee et al. 2017).

However, the variation in texture, location, size, and shape of breast tumors present practical challenges, necessitating the improvement of the overall performance of CAD systems to minimize false positives and negatives. With the recent developments in computer technology, marked by improved computational power and speed, research have led to a widespread recommendation of deep learning algorithms in biomedical fields, especially in developing CAD systems for mammography (Al-masni et al. 2018; Eltrass and Salama 2020; Ragab et al. 2022).

Over the past two decades, deep learning has shown increasing success in numerous computer vision tasks and has been proven a worthy candidate to overcome challenges in the medical imaging domain. Consequently, a variety of techniques have been suggested and implemented in mammography screening, with most of them focusing on the segmentation of breast lesions, tumor detection, and classification (Hamed 2020; Abdelhafiz et al. 2020; Jiao et al. 2017).

Recently, deep learning models have gone past the simple adaptation of the convolutional neural networks (CNN) used in earlier stages of computer vision tasks,

and several newer and more sophisticated CNN architectures have outperformed image classification results. The CNN model was initially developed for image classification tasks and has been the groundwork for newer state-of-the-art architectures such as ResNet, VGG, Xception, GoogleNet, and AlexNet. As a result, numerous studies have used these architectures for the classification of breast lesions, incorporating them in CAD systems in combination with techniques such as ensemble learning, fusion modeling, and transfer learning (Baccouche et al. 2022).

However, very often, these studies focus just on maximizing the classification performance of their proposed method, thereby neglecting the interpretability of CNNs compared to traditional neural networks. Although deep learning models are frequently described as 'black boxes', meaning the learned representations are difficult to extract and present in human-readable form, CNNs differ in that they represent visual concepts, allowing for the visualization and interpretation of what the model learns and its decision-making process. In the last decade, a wide range of techniques have been proposed to visualize and interpret these visual concepts (Chollet 2017). Apart from interpreting what the model learned, it is also possible to visualize why the model made a certain classification decision. This information can be of great use for radiologists when screening mammograms.

In this study, we conduct the final stage of the traditional CAD system, diagnosing and classifying breast mass mammography images using a variation of the state-of-the-art ResNet architecture called the ResNet50 model. Additionally, we try to interpret the model by using visualization techniques. Consequently, we will show intermediate activation maps, intermediate kernel representations, and class activation maps using Gradient-weighted Class Activation Mapping (Grad-CAM) to show classification decisions. To train and evaluate the performance of the model, a standardized breast mammography database is used called the Curated Breast Imaging Subset of the Digital Database for Screening Mammography (CBIS-DDSM). Moreover, we compare the ResNet50 model's performance against a baseline lasso regularized logistic regression model, interpreting the latter through visual representations of its coefficients.

However, before applying the state-of-the-art techniques for breast lesion classification, we first familiarize ourselves with a simple computer vision task: classifying images of 0's and 1's using a logistic regression and a basic CNN architecture. For this purpose, the Modified National Institute of Standards and Technology (MNIST) database is used. This preliminary phase also introduces the interpretative techniques to be applied later in the study.

The remainder of this thesis is structured as follows: in Section 2, we introduce a literature review regarding the most successful mass classification and diagnosis publications using deep learning, transfer learning, and ensemble learning techniques. Then, in Section 3, we cover the logistic regression model, what method is used to estimate the model, results regarding the classification task, and model interpretation. Additionally, in the same section, we introduce the theory behind typical

CNN models, what architecture is used for the MNIST classification task, the performance of the model, and we interpret the model using the visualization techniques mentioned before. In Section 4, we continue with the main methodology in this study, discussing the CBIS-DDSM dataset and the data preprocessing techniques applied to it, as well as the extension of the logistic regression model to a regularized version of the logistic regression and the simple CNN to the ResNet50 model. Next, in Section 5, we present the results of both models, consisting of performance and interpretation. Then in Section 6, we discuss our methodology and give some recommendations for future research. Finally, in Section 7, we finish the study with our conclusions.

2 Literature review

In recent years, numerous studies have proposed machine learning methodologies within computer-aided diagnosis (CAD) systems, aimed at strengthening physicians' diagnostic understanding for breast mass differentiation in mammography. In this context, by Dhahri et al. (2020) used a Tabu search algorithm for the identification of the most important features, which were then analyzed using a K-Nearest Neighbors (KNN) algorithm to classify breast lesions as either malignant or benign.

Since the development of deep learning, more attention has been given into the integration of deep learning models into CAD systems because of their better efficiency over traditional systems that rely heavily on manual feature extraction. One end-to-end methodology introduced by Shen et al. (2019), classified mammograms into categories of cancerous or non-cancerous using an advanced CNN architecture, incorporating both VGG and ResNet networks. This approach resulted in an area under the receiver operating characteristic curve (AUC) of 0.91 when evaluated on the CBIS-DDSM dataset. Similarly, another end-to-end model named DiaGRAM, developed by Shams et al. (2018), merged a CNN with Generative Adversarial Networks (GAN) for the classification of mammograms, achieving an accuracy of 89% on the DDSM dataset, an older and less accessible version of CBIS-DDSM.

Further advancements were seen with the introduction of a novel approach that integrated texton features with deep CNN features by Zhang et al. (2020). They got to an accuracy of 94.30% on the CBIS-DDSM dataset. Another significant contribution by Muramatsu et al. (2020) involved the enhancement of a CNN model's accuracy through the incorporation of synthetic data derived from lung nodules in computed tomography (CT) scans, utilizing cycle GANs, which manifested an accuracy of 81.4% on the DDSM dataset.

A groundbreaking method proposed by Chakravarthy and Rajaguru (2021) merged deep learning techniques with an extreme learning machine (ELM) for the classification of abnormal regions of interest (ROI) images into malignant or benign categories, achieving remarkable accuracies of 97.19% on DDSM, In a subsequent study, Nasir Khan et al. (2019) employed a multi-view feature fusion (MVFF) based-CAD

system to enhance CNN performance by integrating data from four mammogram views, facilitating the classification into malignant or benign with an AUC of 0.84 on the CBIS-DDSM database.

Beyond the use of ensemble learning techniques, the concept of transfer learning has been widely incorporated with deep learning strategies to classify benign and malignant breast tumors. In this regard, Alkhaleefah et al. (2020) employed a dual-shot transfer learning (DSTL) approach, entailing the fine-tuning of various pre-trained networks initially on an ImageNet dataset, subsequently on a similar larger dataset, followed by the target dataset. This methodology, when applied to the CBIS-DDSM, surpassed the efficacy of traditional single-shot transfer learning methods, manifesting an average AUC of 0.91 on the CBIS-DDSM dataset. More recently, Lee et al. (2017), incorporated transfer learning with a stacked ensemble of three different ResNet networks, attaining an accuracy of 0.87 on the CBIS-DDSM.

In an effort to improve mass classification and diagnosis, researchers have highlighted the importance of defining the textural and morphological characteristics of breast tumors to help radiologists during their assessment. Bi et al. (2019), for example, demonstrated a strong correlation between the likelihood of malignancy and the morphological attributes of a breast lesion. Consequently, numerous studies have integrated a segmentation stage in their CAD system to improve diagnosis. Tsochatzidis et al. (2021) enhanced the diagnostic accuracy of breast cancer by modifying convolutional layers of a CNN to process both input images and their corresponding segmentation maps, attaining AUC values of 0.89 and 0.86 on the DDSM and CBIS-DDSM datasets, respectively. Additionally, Ragab et al. (2019) introduced region based and threshold segmentation techniques, combined with a pre-trained AlexNet network, achieving an accuracy of 0.71 and 0.74 for the DDSM and CBIS-DDSM datasets, respectively.

The development of CAD systems towards fully automated breast cancer diagnosis procedures, which analyze the entire mammogram images and produce a final diagnosis, represent a significant step forward. Singh et al. (2019) presented an automated workflow detecting breast tumor regions from mammograms using the Single Shot Detector (SSD), followed by outlining its segmented mask with a conditional Generative Adversarial Network (cGAN), culminating in shape classification via a CNN, with the framework reaching an overall accuracy of 80% for shape classification. In a similar study, Al-antari Aisslab et al. (2018) suggested a fully integrated CAD system employing deep learning techniques for with mass detection using the You-Only Look Once (YOLO) architecture, mass segmentation using a Full resolution convolutional network (FrCN), and concluding with the classification of detected and segmented masses into benign or malignant using a CNN model.

Now that we have covered some of the most successful literature around breast cancer detection in detail, we will continue with the suggested methodology of this study. More specifically, we first start with the preliminary methodology.

3 Preliminary Methodology

In this study, we propose the use of the well known ResNet50 CNN architecture for the classification task of predicting malignant or benign cases in mass breast mammography images, and compare our results to a regularized logistic regression baseline model. Additionally, we perform techniques to visualize and interpret the model by showing the representations learned by the network. For the classification part, the methodology employs different strategies: transfer learning, fine-tuning, image enhancement, and image augmentation. For the second part, where we try to interpret the model, we focus on visualizing intermediate activations of convolutional layers, visualizing the networks parameters (or kernels), and visualizing heatmaps of class activation.

However, before going into the complex and deep CNN architecture that ResNet50 is, and the complex breast mammography images the network has to learn from, we first start simple using the MNIST dataset for the classification task of predicting digits in images. For this task, we use a logistic regression and a simple CNN architecture, and we try to understand what is happening in both models.

3.1 MNIST Dataset

The MNIST dataset contains 70,000 images of handwritten digits from 0 through 9. The images are grayscale and have a size of 28×28 pixels. The images are visual representations of 28×28 matrices with each entry having a value between 0 and 255, with 0 representing a black pixel and 255 representing a white pixel. The primary task associated with the dataset is to classify a given image of a digit into one of the 10 classes. However, since in this study the ultimate focus is a binary classification task, we filter out digits 2 through 9 and only focus on the 0's and 1's. In Fig. 1, we show a selection of 0's and 1's. Although the 0's all have very similar shape in the dataset, 1's can occasionally be written with the notation used in the second 1-image of the figure. Filtering out digits 2 through 9 leaves us with a total of 14780 images: 6903 times 0 and 7877 times 1. For the purpose of training and evaluating the models, we assign 85% of the images to a training set and 15% to a test set while keeping the original class balance.

In the next sections, we will first discuss the logistic regression model, its theoretical properties, and the method used to estimate the model. We will then use the model to classify the selection of the MNIST dataset and look at some results. Afterwards, using a similar approach, we will introduce the CNN model, use it for classification, and try to understand it by visualizing the representations learned by the simple network.

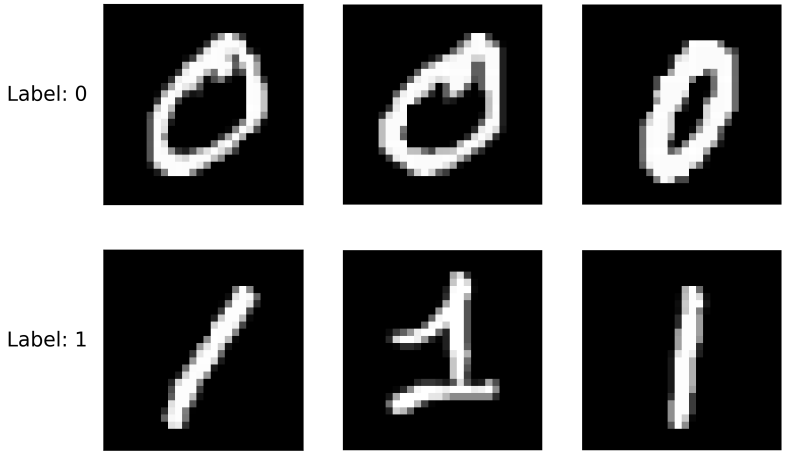


Figure 1: Samples of the MNIST dataset images containing the digits 0 and 1. The 0 digits consistently have similar shapes, while the 1 digit can occasionally been written with different notation.

3.2 Logistic Regression

A linear regression is typically used to model the relationship between a response variable Y_i and one or more explanatory variables X_{ij}. The goal is to find a linear equation that best predicts the response from the explanatory variables:

$$Y_i = \beta_0 + \sum_{j=1}^{d} \beta_j X_{ij} + \epsilon_i, \tag{1}$$

where β_0 is the intercept, β_j the coefficients, and ϵ_i the residual. When the response is of binary nature (i.e., $Y_i \in \{0, 1\}$) rather than continuous, the regression function corresponds to the probability that the response equals 1 given X, where $X = (X_1, \ldots, X_d)$. This interpretation follows from two facts. First, the regression function is the expected value of Y for a given X: $\mathbb{E}(Y|X_1, \ldots, X_d)$, second, if Y is binary, then the expected value is the probability that $Y = 1$: $\mathbb{E}(Y) = 0 \cdot \Pr(Y = 0) + 1 \cdot \Pr(Y = 1) = \Pr(Y = 1)$ (Stock and Watson 2019). Consequently, a coefficient β_j of the regressor X_j is the change in probability that $Y = 1$ given a unit change in X:

$$\Pr(Y = 1|X_1, \ldots, X_d) = \beta_0 + \beta_1 X_1 + \cdots + \beta_d X_d. \tag{2}$$

However, since the regression function is a continuous function, being able to exceed 0 and 1 (thus exceeding probabilities of 0% and 100%), it makes sense to use some nonlinear link function F that maps the outcome of the regression function between 0 and 1. Common choices for such functions are cumulative probability distribution functions (c.d.f.). For logistic regression, the logistic c.d.f. is used,

resulting in the following model:

$$\begin{aligned}\Pr(Y = 1|X_1, \ldots, X_d) &= F(\beta_0 + \beta_1 X_1 + \cdots + \beta_p X_d) \\ &= \frac{\exp(\beta_0 + \beta_1 X_1 + \cdots + \beta_p X_d)}{1 + \exp(\beta_0 + \beta_1 X_1 + \cdots + \beta_d X_d)}.\end{aligned} \tag{3}$$

Although for the linear regression function the interpretation of the coefficients for a continuous response is straightforward, when using a logistic regression, it is somewhat trickier. To get an idea, we start with defining the odds that $Y = 1$ for a given X:

$$\begin{aligned}\text{Odds}(Y = 1) &= \frac{\Pr(Y = 1|X_1, \ldots, X_d)}{\Pr(Y = 0|X_1, \ldots, X_d)} \\ &= \frac{\Pr(Y = 1|X_1, \ldots, X_d)}{1 - \Pr(Y = 1|X_1, \ldots, X_d)} \\ &= \frac{\exp(\beta_0 + \beta_1 X_1 + \cdots + \beta_d X_d)}{1 + \exp(\beta_0 + \beta_1 X_1 + \cdots + \beta_d X_d) - \exp(\beta_0 + \beta_1 X_1 + \cdots + \beta_d X_d)} \\ &= \exp(\beta_0 + \beta_1 X_1 + \cdots + \beta_d X_d) \\ \log(\text{Odds}(Y = 1)) &= \beta_0 + \beta_1 X_1 + \cdots + \beta_d X_d.\end{aligned} \tag{4}$$

In the last step, we take the logarithm of the odds to get rid of the exponent on the right-hand side. This result tells us that a coefficient β_j represents the change in the log odds that $Y = 1$ for a unit change in X_j holding all other regressors constant. For positive β_j, an increase in X_j means an increase in the log odds that $Y = 1$, and therefore an increase in the probability that $Y = 1$. The exact magnitude of the change in probability depends on the value of X_j, but the direction of change for positive β_j is towards a higher probability that $Y = 1$. For negative β_j, this is the other way around (James et al. 2013).

To compute the change in probability that $Y = 1$ for a change in X_j, first the predicted value given the original X using the estimated model is computed. Then, the predicted value for an adjusted value of $X = (X_1, \ldots, X_j + \Delta X_j, \ldots, X_d)$ is computed. And finally, the difference between the two predicted values is computed. However, in this study, we only focus on the direction of β_j, and hence the direction of change in the probability that $Y = 1$, to identify which pixels are important for the classification of a 1-image and 0-image. This is explained further in the next section.

To estimate the regression coefficients, a method called maximum likelihood estimation (MLE) can be used. This method computes coefficients such that it maximizes the joint probability distribution of the data, also called the likelihood function. In a more intuitive sense, MLE chooses values for the coefficients that maximize the probability of drawing the observed data. Hence, MLE coefficients are the coefficients that most likely produced the data (Stock and Watson 2019).

Since our response is binary, we have that Y_i is Bernoulli distributed as $Y_i \sim \text{Bernoulli}(P_i)$, where P_i equals Eq. (3) for a given $X_i = (X_{i,1}, \ldots, X_{i,d})$. Therefore, our joint probability distribution or likelihood function becomes

$$f(\beta_0, \beta) = \prod_{i=1}^{n} P_i^{Y_i}(1 - P_i)^{1-Y_i}, \tag{5}$$

where $\beta = (\beta_1, \ldots, \beta_d)$. Since there is no difference between maximizing the likelihood function and its logarithmic form, often a more convenient notation is used, called the log-likelihood function:

$$\log f(\beta_0, \beta) = \sum_{i=1}^{n} Y_i \log(P_i) + \sum_{i=1}^{n} (1 - Y_i) \log(1 - P_i). \tag{6}$$

Now differentiating $\log f(\beta_0, \beta)$ with respect to each β_j gives us d equations. When these d equations are equated to zero, the resulting conditions are called first order conditions. Solving the first order conditions, we find the maximum likelihood estimator $\hat{\beta} = (\hat{\beta}_1, \ldots, \hat{\beta}_d)$. Unfortunately, $\hat{\beta}$ can not be derived in closed form, hence numerical methods are necessary to find the estimates.

In this study, our main focus is how to optimize neural networks for image classification, and only use the logistic regression model as baseline for performance and interpretability. Therefore, we choose not to go into details about the numerical methods to compute the maximum likelihood estimates. Instead, we quickly mention some of the most popular methods used in Python libraries and which one is used in this study.

The first method is called the Limited-memory Broyden–Fletcher–Goldfarb–Shanno (L-BFGS) algorithm, proposed by Liu and Nocedal (1989). This method is particularly well-suited for problems with a large number of coefficients. Then another popular method is the Newton-Conjugate Gradient (Newton-CG) algorithm proposed by Shewchuk (1994), which uses the Newton-Raphson method in combination with a conjugate gradient approach. Finally, a newer method proposed by Defazio et al. (2014) called Stochastic Average Gradient Descent Algorithm (SAGA) can be used, which is a variant of the stochastic gradient descent. In this study, the latter is used since it is a faster algorithm for large sample sizes.

Once the maximum likelihood estimates are computed, we can perform statistical inference. Since the MLE is normally distributed for large sample sizes, statistical inference of the MLE logistic regression coefficients proceeds in the same way as for the Ordinary Least Squares (OLS) estimated linear regression. That is, t-statistics are used for hypothesis testing, 95% confidence intervals are computed using $\pm$ 1.96 standard errors, and joint hypothesis tests on multiple coefficients use the F-statistic. All of this is analogous to the linear regression model.

For the measure of fit however, denoted as the R^2 in the continuous response case, a few changes have to be made. The measure of fit used for binary responses is called the pseudo-R^2. It measures the fit of the model using the likelihood function.

Because the MLE maximizes the likelihood function, and adding regressors will increase the value of the maximized likelihood, we want to measure the improvement in fit compared to a null model. Therefore, the pseudo-R^2 measures the quality of the fit by comparing the maximized likelihood of the estimated model versus a model without any regressors. The former is often called the full model and the latter the null model. The expression of the pseudo-R^2 is as follows:

$$\text{pseudo-}R^2 = 1 - \frac{\log f_{\text{full}}^{\max}}{\log f_{\text{null}}^{\max}}, \tag{7}$$

where $\log f_{\text{full}}^{\max}$ and $\log f_{\text{null}}^{\max}$ are the maximized likelihoods for the full and null model respectively.

While the pseudo-R^2 can be interpreted as a measure of improvement in fit compared to a null model, we need another test to see whether this improvement is statistically significant. To do so, we can use the log-likelihood ratio test (LRT). This test statistic is derived from the difference in log-likelihoods between the two models and follows a χ^2-distribution (with the absolute difference between the amount of regressors of the models as degrees of freedom) under the null hypothesis that the null model is true. The LRT-statistic is given by:

$$\text{LRT} = -2 \log \frac{f_{\text{null}}^{\max}}{f_{\text{full}}^{\max}} \tag{8}$$

Since our goal is not just to do inference on the model itself, but also evaluate the model on out-of-sample predictions, we need an evaluation metric. Although there are many such metrics, in this preliminary phase, we will only use the accuracy. This is a metric that calculates the ratio between the number of correct predicted samples and the total number of evaluation samples, which gives a good performance indication when the classes are balanced. This is the case for our data, since we have 6903 0's and 7877 1's. The accuracy is expressed as follows:

$$\text{accuracy} = \frac{\#\text{ correct predictions}}{\#\text{ samples}} \tag{9}$$

Now that we have covered everything necessary to apply the model, we will continue using it for the classification of 0's and 1's of the MNIST dataset. We will evaluate the model using the pseudo-R^2, compute the LRT-statistic and its p-value, and look at the prediction performance using the accuracy. Since the classification task asks for a large amount of regressors (see next section), we will not perform statistical inference on the coefficients. Instead, we will visualize the coefficients and use the interpretation framework discussed before to interpret the model and its coefficients, thereby following the approach used in Hastie et al. (2015).

During the initial exploratory phase of modeling with logistic regression, we also conducted a Monte Carlo simulation to investigate the performance and consistency of the logistic regression coefficients under controlled conditions. We assessed how well the logistic regression could recover self-determined coefficients when trained

Metric	Value
pseudo-R^2	0.99
LRT-statistic	17495.21
p-value of LRT-statistic	0.00
Accuracy	0.99

Table 1: Logistic regression inference and prediction accuracy results when estimated and evaluated on the MNIST dataset for 0 and 1 digits.

on artificially generated data that mimics the structure of the MNIST dataset. However, to better understand the method and interpretation of the results of the simulation, it is recommended to first read the next section. In Appendix A, the results are shown. From the simulation results, we could conclude that the model was able to consistently recover the self-determined coefficients.

3.2.1 Logistic Regression MNIST Classification

As mentioned in Section 3.1, the MNIST dataset includes 28×28 images that can be represented by matrices with pixel values between 0 and 255. However, the logistic regression model is not build to handle such data dimensions. Therefore, each 28×28 matrix is transformed to a 784-dimensional vector to suit the structure of the model. As a result, we obtain the following logistic regression:

$$\Pr(Y = 1|X_1, \ldots, X_{784}) = \frac{\exp\left(\beta_0 + \sum_{j=1}^{784} \beta_j X_j\right)}{1 + \exp\left(\beta_0 + \sum_{j=1}^{784} \beta_j X_j\right)}. \tag{10}$$

where each explanatory variable X_j represents a pixel in the image. For instance, X_1 corresponds to the top left pixel in the image, and X_{784} corresponds to the bottom right pixel in the image. We estimate and perform inference on the model on the aforementioned 85% of the data and evaluate the performance on the remaining out-of-sample 15% of the data. Table 1 shows the logistic regression results.

We observe that the pseudo-R^2 has a value of 0.99, indicating that the model explains nearly all the variability of the response variable with the regressors used. The LRT-statistic of 17495.21 is large as well, having an associated p-value of 0.00. This means we reject the null hypothesis that the null model is true, and accept the alternative that the model with the regressors provides a significantly better fit to the data, which is consistent with the high pseudo-R^2 value. Additionally, the accuracy of 0.99 means that the model correctly predicts the class of the digit 99% of the time on the out-of-sample test data. This is a very good accuracy rate and suggests that the model performs very well in classifying the digits.

After estimation and inference, we encounter the challenge of interpreting the large amount of coefficients (784 in total). As mentioned before, regressor X_1 corresponds to the top left pixel in the image, hence coefficient β_1 represents the change

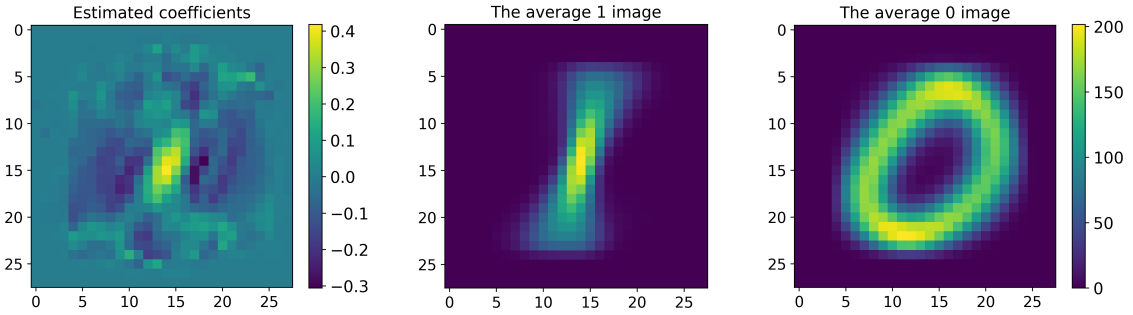


Figure 2: (Left panel): The 784-dimensional coefficient vector of the logistic regression in Eq. (10) displayed as a 28×28 heatmap. The coefficients are estimated using 0-images and 1-images of the MNIST dataset in combination with MLE. (Middle & right panel): The average 1 and 0 image of the MNIST dataset.

in the probability that $Y = 1$ for a unit change in the top left pixel value. For positive β_1, this mean an increase in X_1 results in a higher probability that the image is a 1, and a negative β_1 means an increase in X_1 results in a higher probability that the image is a 0.

Following this logic for each coefficient-regressor pair, if we transform our 784-dimensional coefficient vector $\hat{\beta}$ to a 28×28 matrix (hence β_{100} is located where the 100th pixel would be located), and plot this matrix as a heatmap, we can identify what parts of the MNIST images are important for the model for 0-classification and for 1-classification. For 0-classification, the coefficients in the heatmap should be negative, and for 1-classification, the coefficients should be positive. To identify these parts more easily, we also show the average 0-image and 1-image computed from the dataset next to the coefficients. The results are shown in Fig. 2.

If we look at the average-images, it is clear that some pixel locations are more important for 1-images than for 0-images. For instance, where 1-images have large pixel values right in the middle, 0-images have very low or zero valued pixels. On the other hand, 0-images have large pixel values at the left and right side (in the middle), while 1-images have a lot of zero valued pixels there. Contrary, the upper and lower side of the images (in the middle) are important for both digits, as those pixels have large values for both digits.

This is exactly what we see back in the heatmap of the estimated coefficients. Coefficients located right in the middle all show positive numbers (yellow color), indicating that if a pixel value is large in that area, the probability that $Y = 1$ increases. On the other hand, coefficients located at the left and right side show negative numbers (dark blue color), indicating that if a pixel value is large in that area, the probability that $Y = 0$ increases. For the upper and lower side of the coefficient heatmap, we see a more mixed set of values, some being positive and some negative. This is due to those areas being important for both 0-images and 1-images.

Now that we have used the logistic regression for the MNIST dataset, where

we estimated, evaluated, and interpreted the model, we will continue with our next model: the CNN. We will first start with a motivation on why we need the model and its advantages over the logistic regression, and then, similar to this section, we will cover all theory necessary to understand the model and use it for the MNIST classification task.

3.3 Convolutional Neural Network

Neural networks (NNs) typically are highly parameterized models, build to mirror the architecture of the human brain. The central idea of NNs is to use linear combinations of given inputs, and then model the output as a nonlinear function of these inputs. Although the econometric toolkit already has many capable tools such as the logistic regression, NNs are a worthy addition because of their versatility and because they are universal function approximators. This is proven by the universal approximation theorem first introduced by Cybenko (1989), and further generalized by Hornik (1991). The theorem states that for any continuous function $f : \mathbb{R}^m \mapsto \mathbb{R}^n$, there exist a NN with one hidden layer, H, such that it can approximate f to an arbitrary level of accuracy. In other words, $|f(\mathbf{x}) - H(\mathbf{x})| < \epsilon$ for any $\epsilon > 0$.

While NNs are not the only universal function approximators, NNs have strong generalization capabilities, scale well to the dimensions of the input space, and rely minimally on assumptions. Such features are not consistently matched by other approaches, and so they result in a powerful learning method which has been proven to work in a wide variety of fields such as finance, art production, novel writing, and most important for our case: image classification tasks (see Dixon et al. (2016); Li et al. (2019); Santos et al. (2021)).

The image data in the MNIST dataset contains relatively small images, with a clear distinction between the two classes, each class showing similar pixel intensities and similar digit shapes. These characteristics make that the logistic regression used in the previous section is adequate to perform the classification task. However, typical image data contains a much higher dimensionality (hence more features), more complex spatial relationships, wider variety per class, different pixel intensities per class, and more contextual information, resulting in challenges that traditional models like logistic regression struggle to overcome.

CNNs, on the other hand, are specifically designed for the challenges of image data. Firstly, their architecture allows for the preservation of spatial information of the high dimensional nature of image data, enabling them to learn from the data efficiently. They do so while using fewer parameters than traditional neural networks by using two principles called sparse connectivity and parameter sharing (Goodfellow et al. 2016). Additionally, the structure of the CNN, including convolutional layers, activation layers, pooling layers, and fully connected layers, facilitate the model's ability to extract and learn from features at different levels of abstractness. For instance, the model is able to learn simple directional edges but also complete objects

within an image.

Although the shift from logistic regression to CNNs might not be necessary for the MNIST images, it is reasonable to assume that it is necessary for our ultimate focus of the complex breast mammography images. These images show a large portion of the challenges mentioned before, and hence the shift is not just a step-up in model complexity, but a necessity for accurate predictions of benign and malignant lesions, due to the nature of the data. Therefore, by using CNNs, we try to overcome the presumably insufficient pattern recognition capabilities of the logistic regression.

However, to familiarize ourselves with the complexity of CNNs, it is good practice to first use it for the MNIST classification task. Before doing so, we will first discuss the aforementioned four distinct layers seen in CNNs, followed by how the model learns, and we finish with some challenges seen while training neural networks. Contrary to logistic regression, since most literature on neural networks is written from a machine learning perspective instead of a pure econometric perspective, this study mostly uses the former perspective and translates some of the ideas discussed to econometric terminology.

3.3.1 Convolutional Layer

Given a 2-dimensional input image $\mathbf{V}$ with elements $V_{j,k}$ representing the pixel values of the image at row j and column k, and a 2-dimensional kernel $\mathbf{K}$ with elements $K_{m,n}$ representing the parameters (equal to coefficients in the logistic regression model) at row m and column n, the output $\mathbf{Z}$ by convolving $\mathbf{K}$ across $\mathbf{V}$ can be expressed as

$$Z_{i,k} = \sum_{m,n} V_{i+m-1,k+n-1} K_{m.n}. \tag{11}$$

In Fig. 3, we illustrate the convolutional operation. To convolve $\mathbf{K}$ across $\mathbf{V}$ means sequentially taking dot products between the kernel and an equally sized image patch by moving along the image dimensions. The output, also called the activation map, is then another 2-dimensional map of linear combinations called activations. The size of the activation map is dependent on the image size, the kernel size, and the step size s used to move along the image. Using s in the convolutional operation, the output translates to

$$Z_{i,k} = c(\mathbf{K}, \mathbf{V}, s)_{i,k} = \sum_{m,n} [V_{(i-1)\times s+m,(k-1)\times s+n} K_{m,n}], \tag{12}$$

where the step size s is called the stride (Goodfellow et al. 2016).

In the illustration, the dimensions of the input image and the kernel size are 4×3 and 2×2 respectively. Using a stride of 1, the output results in an activation map of size 3×2, which is smaller than the input. When using a CNN with a large amount of convolutional layers, each using the output of the previous layer as their input, the spatial extent of the network can rapidly reduce, limiting the ability of the network to learn important features of the image (Goodfellow et al. 2016). Therefore, a

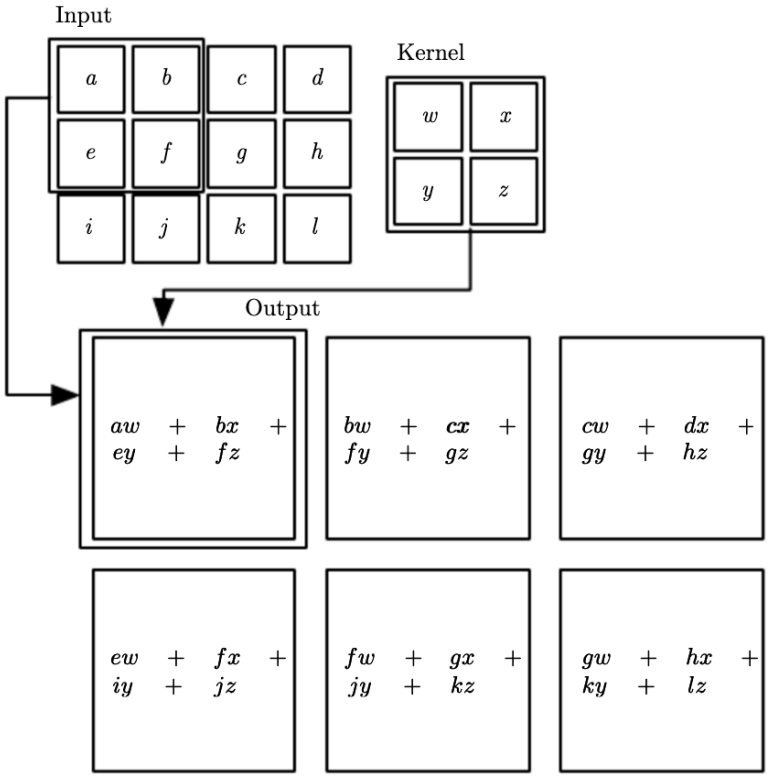


Figure 3: The convolutional operation seen in convolutional layers for a 4×3 input image, a 2×2 kernel, and a stride s of 1, resulting in a 3×2 activation map with units that are linear combinations of the kernel and equally sized image patches of the input. More specifically, the dot product between the 2×2 image patch in the top left of the input and the kernel is taken, resulting in a linear combination of $aw + bx + ey + fz$. Then the image patch is moved along the horizontal axis of the image and another dot product is taken. After covering the image horizontally, the image patch is moved vertically one step and the horizontal steps are taken again.

feature called zero-padding is used, which allows inserting 0 valued entries around the input (in any of the layers) to increase its dimensions. This allows us to control the kernel and output size independently and does not decrease the learning ability of the model by assuming no relevant information of the image is located at the edge of the image. The size of the output of the convolutional layer can be calculated using the following expression (Ragab et al. 2019):

$$\text{Output size of convolutional layer} = \frac{\text{Input size} - \text{Kernel size}}{\text{Stride}} + 1 \tag{13}$$

Often, kernels respond or activate one particular pattern in the input, such as vertical edges seen in the images of the digit 1 (Efron and Hastie 2016). However, we want the convolutional layer to learn a lot more than just one pattern, especially when dealing with complex images such as breast mammography. Therefore, in each convolutional layer, multiple kernels are used that each convolve completely over the input. After convolving all kernels, each resulting activation map is gathered in a 3-dimensional activation map. For instance, using two instead of one kernel in the illustration, the resulting activation map would have size $3\times2\times2$.

Now both the sparse connectivity and parameter sharing principles mentioned before already become apparent in the convolutional operation. In traditional neural networks, each layer is fully connected to the next layer, meaning each unit in the output would not be a linear combination of just an image patch and a kernel, but a linear combination of all the units of the full image and an equally sized parameter vector that has much more entries than our kernel. Hence, by layers not being fully connected but sparsely connected, the amount of parameters is drastically reduced.

Additionally, while in the convolutional operation the same kernel is used while convolving over the input and calculating the units of the activation map, in traditional neural networks, for each unit of the output, a new parameter vector is used. Hence, keeping the same kernel for calculating the units of the activation map reduces the amount of parameters used even more.

So far, the convolutional operation we discussed is nothing more than what standard linear models use seen in basic econometrics. That is, a response (one unit of the activation map) being equivalent to a linear combination of some regressors (a small selection of the pixels), each multiplied by a parameter (the kernel parameters):

$$u_{11} = \beta_1 X_{11} + \beta_2 X_{21} + \beta_3 X_{12} + \beta_4 X_{22}, \tag{14}$$

where u_{11} is the first unit of the activation map, β_i are the parameters of the kernel, and X_{jk} are the regressors or pixel values at position j, k. In fact, if a CNN were only to consist of convolutional layers, the model would reduce to a generalized linear model (GLM) (Efron and Hastie 2016). The CNN becomes a nonlinear generalization of the linear model only due to the use of nonlinear transformations seen in the activation layer. By introducing these nonlinear transformations, we greatly enlarge the class of linear models.

3.3.2 Activation Layer

The activation layers (or nonlinear transformations) are always present after a convolutional layer. Each activation unit in each activation map is transformed using a nonlinear activation function. Popular choices for these functions are the sigmoid, rectified linear unit, hyperbolic tangent, or leaky rectified linear unit function (Nair and Hinton 2010; Maas 2013; Olgac and Karlik 2011).

In this study, the rectified linear unit (ReLU) function is used, as this is recommended and nowadays the standard when working with neural networks (Goodfellow et al. 2016; Fuleky 2019). The function keeps the output the same value unless it is smaller than 0. In that case, it is mapped to 0:

$$f(x) = \begin{cases} x, & \text{if } x > 0 \\ 0, & \text{otherwise.} \end{cases} \tag{15}$$

The nonlinear activation function is similar to a link function used in logistic regression. Although they are used for different purposes, both transform the response using some user-specified function. For instance, where in the logistic regression we transformed $\Pr(Y = 1|X_1, \ldots, X_d) = \beta_0 + \sum_{j=1}^{d} \beta_j X_j$ to $\Pr(Y = 1|X_1, \ldots, X_d) = F(\beta_0 + \sum_{j=1}^{d} \beta_j X_j)$ to map the response to the interval (0,1) using the logistic c.d.f. (see Eq. (3)), in our CNN, each unit in the activation map is transformed from $u_{11} = \beta_1 X_{11} + \beta_2 X_{21} + \beta_3 X_{12} + \beta_4 X_{22}$ to $u_{11} = f(\beta_1 X_{11} + \beta_2 X_{21} + \beta_3 X_{12} + \beta_4 X_{22})$ to map the units to the interval $[0, \infty)$ using the ReLU function. Since the activation function is used for numerous units in numerous activation maps, the CNN becomes highly nonlinear.

3.3.3 Pooling Layer

The pooling layer always comes after a convolutional layer and an activation layer, and replaces nearby units of the input with a summary statistic. For instance, a pooling layer using max pooling would take the maximum value of a rectangular neighborhood of units and remove all other units. In Fig. 4, we illustrate the max pooling operation for an input of 4×4, a filter of 2×2 which defines the neighborhood of the units, and a stride of 2. Other popular pooling functions include average pooling and L^2 norm pooling, both being applied to the neighborhood defined by the filter (Goodfellow et al. 2016).

Pooling helps to make the representations become invariant to translations of the input. This means that when we translate or shove the input by a small amount, the values of most of the pooled outputs do not change. This property can be very useful if we care less about the location of a feature in the image, but more about whether that feature is present in the image (Efron and Hastie 2016).

The size of the output of the pooling layer is just like the convolutional layer dependent on the input size, the filter size, and the stride. This is similar to Eq. (13),

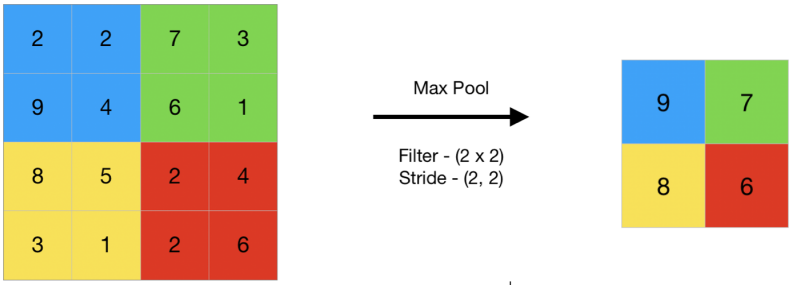


Figure 4: The max pooling operation seen in the pooling layer. Given a 4×4 input image, a 2×2 filter, and a stride of 2, the max pooling operation takes the maximum value of the 2×2 filter area and an equally sized image patch. For instance, for the 2×2 blue area of the input image (the image patch), the maximum value is 9. From this image patch, only this value is taken and the rest is neglected in the output.

the only difference is that we now have a filter only used to define the rectangular neighborhood.

3.3.4 Fully Connected Layer

After some iterations of the first three distinctive layers are applied, the network's output is still a 3-dimensional object. Therefore, to predict classes, the output is flattened to a high dimensional vector and reduced in dimension using fully connected layers seen in traditional neural networks. Similar to the convolutional layers, except for the last one, each fully connected layer is followed by an activation layer with nonlinear activation function. The last fully connected layer has the same dimensions as the amount of classes in the classification task. Consequently, for our task of classifying two classes, the output layer is a 2-dimensional vector.

In Fig. 5, we illustrate a full CNN with one iteration of the first three distinctive layers, a flatten operation, and two fully connected layers to reduce the dimensions for our classification task. Each layer shows the dimensions, starting from a 28×28 MNIST image to an output vector with dimension 2.

Later when we incorporate this CNN for the classification of the MNIST dataset, we will discuss the kernel and filter sizes used, as well as the stride for convolving the images. First we discuss the loss function used for the classification task and how we can use it to train the model using a method called backpropagation.

3.4 Loss Function

The loss function used for our task is called the categorical crossentropy loss function, also known as the softmax loss. It is commonly used in classification tasks where an instance can belong to one of many possible classes. The loss function is particularly suitable for multi-class classification problems where the classes are mutually exclusive. This is the case when having a classifier with two output units

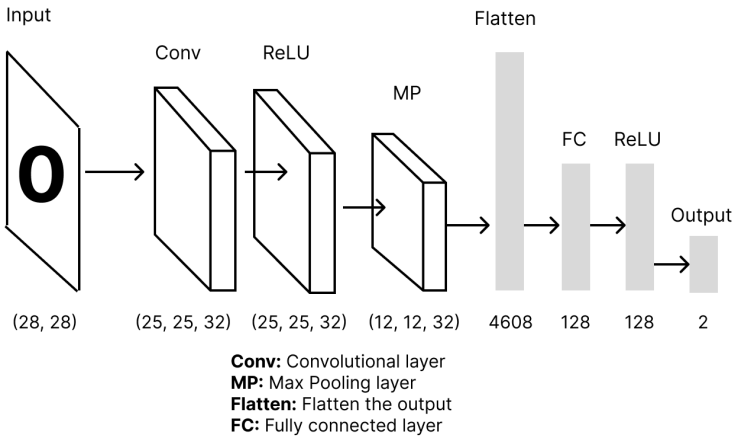


Figure 5: A simple CNN architecture for the classification of 28×28 0-images and 1-images of the MNIST dataset. The network contains a convolutional layer with 32 4×4 kernels with stride 1, a ReLU activation layer, a max pooling layer with filter size 2×2 and stride 2, a flatten operation, and fully connected layers to reduce the dimensions to an output vector with dimension 2.

corresponding to two classes. The function measures the distance between two probability distributions: the true distribution (the class labels) and the predicted distribution (the output of the classifier). The idea is to optimize the difference between the two distributions (Fuleky 2019).

For a single image with label vector $\mathbf{y}$ and predicted probability vector $\mathbf{p}$, the categorical crossentropy loss function is defined as:

$$L(\mathbf{y}, \mathbf{p}) = -\sum_{i=1}^{C} y_i \log(p_i), \tag{16}$$

where C is the number of classes, $\mathbf{y}$ is a one-hot encoded vector of true class labels, with y_i being 1 for the correct class and 0 otherwise, and $\mathbf{p}$ the model's predicted probabilities for each class, with p_i being the predicted probability for class i.

For a binary classification problem, where $C = 2$, and our true class vector labels are [0,1] and [1,0], the loss function reduces to:

$$L(\mathbf{y}, \mathbf{p}) = -(y_1 \log(p_1) + (1 - y_1) \log(1 - p_1)), \tag{17}$$

where y_1 is the first element of the true class vector $\mathbf{y}$, and p_1 is the corresponding predicted probability. As the values of the output vector of the model do not come out as probabilities, a softmax function is applied that converts the raw predictions to probabilities that sum to one. The softmax function for our binary case is defined as:

$$p_i = \frac{\exp z_i}{\exp(z_1) + \exp(z_2)}, \tag{18}$$

where z_i are the raw predictions of the model, which can be interpreted as linear combination of the input features through the neural network's architecture, subjected to nonlinear transformations. Hence, the raw predictions in the output vector

of the model are unbounded and can take any real value (i.e., $z_i \in (-\infty, \infty)$), and are then mapped to probabilities (i.e., $p_i \in [0, 1)$) using the softmax function, where the sum of p_i's is equal to one.

To put things into econometric perspective, we observe that the loss function in Eq. (17) is exactly the one used for our logistic regression, and that the softmax function simplifies to the logistic function used in the logistic regression (see Section 3.2). Consequently, the categorical crossentropy loss function we optimize the CNN over, is actually the log-likelihood function used in the logistic regression model. By minimizing this loss, our CNN model is performing a process identical to MLE. Therefore, the estimated parameters of the CNN model are the parameters that maximize the probability of drawing the observed data and are consistent for large sample sizes. Similarly to before, the parameter estimates cannot be derived in closed form, but the loss function is differentiable. This allows us to use gradient-based optimization to solve the minimization problem. In the next section, we will discuss how this works using a method called backpropagation.

3.5 Gradient Descent & Backpropagation

Gradient descent is an optimization algorithm that is used to minimize some loss function (i.e., minimizing the difference between the actual and predicted value). The goal of the algorithm is to find a set of parameters that result in the minimum possible value of the loss function. This process is analogous to descending a hill: for every step taken, the algorithm calculates the gradient of the loss function, and a step with steepest descent is taken. This is done by taking the average gradient calculated for all observations n. Since the gradient takes a step of steepest ascent, the negative of the gradient is taken for a steepest descent. Hence, iteratively, the algorithm tries to find the lowest minima in the loss function landscape corresponding to the optimal solution (Efron and Hastie 2016). Mathematically, each iteration can be expressed as:

$$\boldsymbol{\theta}_{\text{new}} = \boldsymbol{\theta}_{\text{old}} - \frac{\eta}{n} \cdot \sum_{i=1}^{n} \nabla_{\boldsymbol{\theta}} L_i(\boldsymbol{\theta}), \tag{19}$$

where $\boldsymbol{\theta}$ represents the parameters of the model, η the learning rate, a hyperparameter that determines the step size of the descent, n the number of observations, and $\nabla_{\boldsymbol{\theta}} L_i(\boldsymbol{\theta})$ denotes the gradient of the loss function $L_i(\boldsymbol{\theta})$ for observation i.

Backpropagation is a method to calculate the gradient of the loss function with respect to the network's parameters. It uses the chain rule to compute these gradients systematically. It starts at the final layer of the network, calculating the error and then propagating the error backwards through the layers. At each layer, the gradient of the loss function is computed with respect to the output of the layer. This information is then used to compute the gradient with respect to the parameters of the layer (Efron and Hastie 2016).

Now suppose we want to train a CNN with loss function $L(\mathbf{V}, \mathbf{K})$, kernels $\mathbf{K}$

applied to an image $\mathbf{V}$ and stride s as defined by $c(\mathbf{K}, \mathbf{V}, s)$ (see Eq. (12)). During forward propagation, the network uses $c(\mathbf{K}, \mathbf{V}, s)$ to output $\mathbf{Z}$, which is then iteratively propagated through the rest of the network and used to calculate $L(\mathbf{V}, \mathbf{K})$. During backward propagation, we compute the gradient of the loss function with respect to the outputs $\mathbf{Z}$, and iteratively propagate these gradients through the network to update the parameters in $\mathbf{K}$. We then receive a gradient tensor $\mathbf{G}$ such that:

$$G_{i,j} = \frac{\partial}{\partial Z_{i,j}} L(\mathbf{V}, \mathbf{K}), \tag{20}$$

where the elements of $G_{i,j}$ correspond to the partial derivative of the loss function with respect to $Z_{i,j}$. To update the parameters in $\mathbf{K}$, we compute the derivative of the loss function with respect to the parameters:

$$g(\mathbf{V}, s)_{i,j} = \frac{\partial L(\mathbf{V}, \mathbf{K})}{\partial K_{i,j}} = \sum_{m,n} G_{m,n} V_{(m-1)\times s+i,(n-1)\times s+j}. \tag{21}$$

If the layer is not the bottom layer of the network, we will also need to compute the gradient with respect to $\mathbf{V}$ to backpropagate the error further down. For this calculation, the following expression is used:

$$h(\mathbf{K}, \mathbf{G}, s)_{i,j} = \frac{\partial L(\mathbf{V}, \mathbf{K})}{\partial V_{i,j}} = \sum_{l,m} \sum_{q} K_{l,m} G_{q,(i-l)\times s+m,(j-m)\times s+q}. \tag{22}$$

In this expression, we are distributing the error back from the output to the locations in the input image that were responsible for creating the activation $Z_{i,j}$. The gradients g and h are then used to perform a step of gradient descent to update the parameters in the kernels (Goodfellow et al. 2016). Combining the notation for gradient descent for CNNs with Eq. (19), we get

$$\mathbf{K}_{\text{new}} = \mathbf{K}_{\text{old}} - \frac{\eta}{n} \cdot \sum_{i=1}^{n} g(\mathbf{V}_i, s)_{k,l} \tag{23}$$

3.6 Adam

While gradient descent is the groundwork of most of the optimization algorithms for deep learning, it is rarely still used. Instead, variants of gradient descent are used such as stochastic gradient descent (SGD), SGD with momentum, and Adam. In this study, the latter is used, which is a variant of a combination of SGD and SGD with momentum. Here we briefly discuss the principles of SGD and SGD with momentum and then cover the Adam algorithm.

Gradient descent is a computationally a very heavy task since it computes gradients over the full dataset. To reduce this computational burden, stochastic gradient descent (SGD) can be used, which is a stochastic approximation of the optimal descent by taking the gradients over a subset m (called a batch) drawn i.i.d. from the

data (Kiefer and Wolfowitz 1952). For SGD, the gradient descent expression in Eq. (23) translates to

$$\mathbf{K}_{\text{new}} = \mathbf{K}_{\text{old}} - \frac{\eta}{m} \cdot \sum_{i=1}^{m} g(\mathbf{V}_i, s)_{k,l} \tag{24}$$

Because SGD computes the gradient over a small batch, more noise is introduced to the optimization process. This can actually help the process to escape local minima in the parameter landscape and lead to faster convergence. However, the trade-off is that the path to convergence is more erratic and the final parameters may be worse than with gradient descent (Goodfellow et al. 2016).

To accelerate the convergence process further, the method of momentum can be used proposed by Polyak (1964). This algorithm introduces a 'velocity' variable $\mathbf{v}$ that represents both the direction and speed of the parameters. The velocity is updated by taking an exponentially weighted moving average of past gradients, which allows the optimizer to build up speed in the directions it is taking, thereby dampening the oscillations in directions where the gradient is noisy or inconsistent (Goodfellow et al. 2016). Incorporating the velocity in SGD, the updating equation becomes:

$$\begin{aligned} \mathbf{K}_{\text{new}} &= \mathbf{K}_{\text{old}} + \mathbf{v}_{\text{new}} \\ \mathbf{v}_{\text{new}} &= \phi \mathbf{v}_{\text{old}} - \frac{\eta}{m} \cdot \sum_{i=1}^{m} g(\mathbf{V}_i, s)_{k,l}, \end{aligned} \tag{25}$$

where ϕ is the parameter determining how much of the past gradients contribute to the current velocity.

The Adam optimization algorithm, named for its use of 'adaptive moments', incorporates the benefits from stochastic approximation and momentum from SGD and SGD with momentum, but in a slightly different manner with additional advantages. For one, it adjusts the learning rate for each parameter individually based on the first and second moments (mean and variance) of the gradients. This adaptability makes the optimizer very effective for models with many parameters. Additionally, it computes exponentially decaying averages of the past gradients and past squared gradients, called the first and second order moments, and uses the square root of these moments to scale the learning rate. In return, this can lead to better convergence of the model. Moreover, to handle the initializations at the first iteration, the Adam algorithm applies bias corrections to both moments to ensure they are more accurate during early iterations (Goodfellow et al. 2016). The updating equation for the algorithm is as follows:

$$\begin{aligned} \mathbf{K}_{\text{new}} &= \mathbf{K}_{\text{old}} + \Delta\mathbf{K} \\ \Delta\mathbf{K} &= -\eta \frac{\hat{\mathbf{s}}}{\sqrt{\hat{\mathbf{r}}} + \delta} \end{aligned} \tag{26}$$

with

$$\begin{aligned} \mathbf{s}_{\text{new}} &= \rho_1 \mathbf{s}_{\text{old}} + (1-\rho_1) g \quad &\rightarrow \quad &\text{Updated biased first moment} \\ \hat{\mathbf{s}} &= \frac{\mathbf{s}_{\text{new}}}{1-\rho_1^t} \quad &\rightarrow \quad &\text{Bias-corrected first moment} \end{aligned} \tag{27}$$

and

$$\begin{aligned} \mathbf{r}_{\text{new}} &= \rho_2 \mathbf{r}_{\text{old}} + (1-\rho_2) g \odot g \quad &\rightarrow \quad &\text{Updated biased second moment} \\ \hat{\mathbf{r}} &= \frac{\mathbf{r}_{\text{new}}}{1-\rho_2^t} \quad &\rightarrow \quad &\text{Bias-corrected second moment} \end{aligned} \tag{28}$$

Here we have the decay rates ρ_1 and ρ_2 in $[0,\ 1)$ for the moments (similar to ϕ in Eq. (25)), δ a constant used for numerical stabilization (often 10^{-8}), and t denotes the iteration.

Now that we have covered how to optimize a CNN, we will continue with some of the challenges encountered when optimizing such models.

3.7 Challenges Using Neural Networks

Training neural networks can be a very challenging task. The models are typically overparametrized and the optimization problem is nonconvex and unstable. Unless techniques to resolve these issues are used, the task can be problematic and resulting in poor performance. In this section, we discuss the problems encountered, and the most common techniques to solve them.

3.7.1 Parameter Initialization

While gradient descent and backpropagation can be used for the optimization of parameters within a neural network, the initializations of these parameters' values have to be considered carefully. Bad initializations may lead to the premature saturation of gradients, reducing them to values near zero. When this happens, the associated neuron will produce the same output, despite variations in the input. These neurons are called 'dead neurons', and although a few of them will not affect the model's performance, if the number of dead neurons becomes too large, then gradient descent loses the ability to update earlier layers. As a result, the network will become almost unresponsive to the input (Fuleky 2019).

Additionally, since the loss function is not strictly convex, possessing many local minima, the final solution is dependent on the starting point of descent. In other words, the solution is dependent on the values of the initial parameters. The Adam optimization algorithm helps in resolving this issue by using momentum to get out of local minima. Still, the algorithm might fail, highlighting the importance of the initialization (Fuleky 2019).

In earlier work, the parameters were initialized using i.i.d. random draws from some distribution (like the standard normal distribution). This is sufficient for small

networks, but inadequate when using larger networks, since the aforementioned problems are more present for those. Bad initializations were even the reason that research believed deep neural networks performed worse than simpler ones (Bengio et al. 2007).

In this study, to solve the issues, a widely adopted method called Glorot initialization proposed by Glorot and Bengio (2010) is used. They suggested that the vanishing gradient problem can be partially controlled by keeping variances consistent across layers. This can be realized by initializing parameters such that the variance of the output of each layer is roughly consistent with the variance of the output of the layer before. To achieve this, all parameters $\beta_i \in \mathbf{K}_p$ are initialized using

$$\beta_i \sim N\left(0, \frac{2}{u_p + u_{p-1}}\right), \tag{29}$$

where $\mathbf{K}_p$ is the set of kernels (with parameters) in layer p, and u_p is the number of output units for layer p.

3.7.2 Regularization

A problem often encountered with neural networks due to the amount of parameters used for the model is overfitting. Since neural networks are most often used to produce the smallest out-of-sample error, we need to use techniques to regularize the model such that it generalizes well from train to test set. One such technique does not focus on adjusting the model, but rather on adjusting what the model is trained on. Since neural networks rely heavily on the size of the datasets they are trained on in order to generalize well, data augmentation can be used. This is a technique that inflates the dataset size and can be seen as a regularization measure (Goodfellow et al. 2016). There are different approaches on how to augment data. Later in Section (4.3.2), we discuss some of these approaches and the technique used in this study.

Another regularization technique introduces the l_2 norm to penalize the loss function. In statistics, this l_2 regularization is known as ridge regression, while in machine learning communities the penalty is called parameter decay. Using the l_2 norm for CNNs, the loss function $L(\mathbf{V}, \mathbf{K})$ becomes

$$L_{l_2}(\mathbf{V}, \mathbf{K}) = L(\mathbf{V}, \mathbf{K}) + \frac{\alpha_p}{2} \sum_{p}^{P} ||\mathbf{K}_p||_2^2, \tag{30}$$

where α_p is the penalty strength for layer p. The consequence of using the penalty term is that for a given nonzero parameter $\beta_i \in \mathbf{K}$, if the gradient descent does not encourage moving in the direction of β_i, the parameter will decay to zero. Therefore, the neural network becomes more sparse and emphasizes the parameters that have significant contribution to reduction of the loss function (Fuleky 2019).

Additionally, Srivastava et al. (2014) proposed a technique to regularize the process, called dropout. The idea behind dropout is simple: during the training process,

units of a specific layer are 'dropped' by setting them to zero with a probability ϕ, while the remaining ones are inflated by a factor of $1/(1-\phi)$. This means that during a particular iteration of the process, the dropped units will not participate in the forward and backward pass through the network. The inflation of the remaining units is done to keep the expected value of the output approximately the same as it would be without dropout, helping to stabilize the learning.

Dropout helps the overfitting problem by breaking heavily correlated updates of connected units. These updates become correlated if units start to compensate for the output of another connected unit. By introducing instability to the inputs of a layer, the units of a layer do not become overly dependent on the units of the previous layer.

Moreover, dropout allows us to approximate many models at once. By setting units (briefly) to zero, the adjusted model can be considered as a sparse version of the complete network. Therefore, Srivastava et al. (2014) suggests that using dropout provides parameter estimates that approximate a model averaging over many sparse networks.

The last technique discussed in this study is called early stopping. The idea of early stopping is to use an independent validation set to evaluate the model on after every training epoch. If for a certain consecutive amount of epochs, the performance of the model on the validation set did not increase, the training procedure is stopped and the parameters of the epoch with the best validation performance are stored. The amount of epochs to wait for improvement is called the patience. Using early stopping can prevent the model overfitting the training data while not generalizing to an independent set.

3.7.3 Scaling of Input

Neural networks do not strictly require detailed assumptions about the data generating process. However, in practice, the performance of neural networks is quite sensitive to certain characteristics of the data. When a model processes multiple features, it's critical that these features are scaled to be relatively similar (ideally within an order of magnitude of each other). In principle, a neural network is designed to adjust to inputs of different scales, but in the early stages of training, inputs with larger scales can overshadow the training process, influencing the adjustment of parameters more heavily. This imbalance can either cause neurons to become unresponsive early on or lead to slow progress in improving the model. Adjusting the scale of inputs before training (by normalizing them to a standard distribution or scaling them to fit within a range of 0 to 1) helps avoid these problems. This step ensures that all inputs contribute more equally to the training process, facilitating smoother and more efficient model training (Efron and Hastie 2016).

Now that we have covered the theory about the CNN model, we will continue with how we can use it for the MNIST classification task using a simple architecture. Afterwards, we try to understand what the model is learning and why it

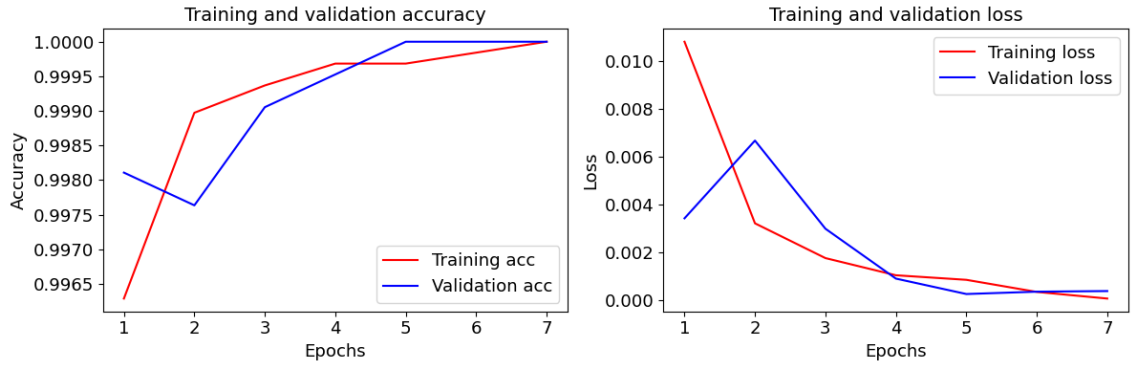


Figure 6: Accuracy (left panel) and loss (right panel) of the CNN model on the training (red) and validation (blue) data of the MNIST dataset after every epoch. During training, early stopping with a patience of 2 was used, resulting in a cancelled training procedure after epoch 7. Both the accuracy of the training and validation set increased to 1, while the loss decreased towards 0.

made its classification decisions by showing the intermediate activation maps of a convolutional layer, the networks kernels, and heatmaps of class activations.

3.8 CNN MNIST Classification

For the classification of 0's and 1's in the MNIST database, we will use the network architecture illustrated in Fig. 5. The network consists of a convolutional layer, a ReLU activation layer, a max pooling layer, and a flatten operation followed by two fully connected layers with a ReLU layer in between. In the convolutional layer, 32 kernels of 4×4 pixels with a stride of 1 are used. For the max pooling layer, a filter size of 2×2 is used, and the output is flattened and reduced to a 2-dimensional output vector necessary for the classification task. The network results in a total of 590754 parameters, which are initialized using Glorot initialization (see Eq. (29)) and optimized using the categorical crossentropy loss function (see Eq. (16)) and Adam optimization (see Section 3.6). Before training, we normalize the data to a range of [0, 1]. During training, we incorporate a batch size of 32 for a total of 10 epochs and use early stopping monitoring with a patience of 2. For every epoch, we store the accuracy (see Eq. (9)) and loss function value for both the training and validation sets. Afterwards, we evaluate the model on the test set using the accuracy as well.

Fig. 6 shows the accuracy and loss progress during training. For both the training and validation set, the accuracy increases gradually over the epochs to 100%, and the loss gradually decreases to a value of $5.8 \cdot 10^{-5}$ and $3.7 \cdot 10^{-4}$ respectively. Moreover, for the test set, the model shows a perfect accuracy score of 100%, outperforming the logistic regression slightly with 1%.

3.9 Intermediate Activations

After training the CNN model, we can analyse what the model learned by visualizing what is happening inside the model. The first step is visualizing the activation maps produced by the convolutional layer and its kernels. This is useful for understanding how the convolutional layer transformed its input, and for getting a first idea of what each kernel is receptive to. Since we used 32 kernels in the convolutional layer, its output consists of 32 activation maps. We show each of these maps for the first 0-image and first 1-image in Fig. 1 by feeding the images to the trained network and extracting the intermediate output. The activation maps are shown in Fig. 7.

From the illustration, we observe that each map is focused on different edges of the image. For instance, the first map of the 0-image is mostly focused on the upper edges of the digit, while the first image of the last row is focused on the lower edges of the digit. Similarly, for the 1-image, some images encode the left edge of the digit and others the right side. Moreover, more clearly than for the 0-image, some activation maps for the 1-image focus on the upper and lower end of the 1.

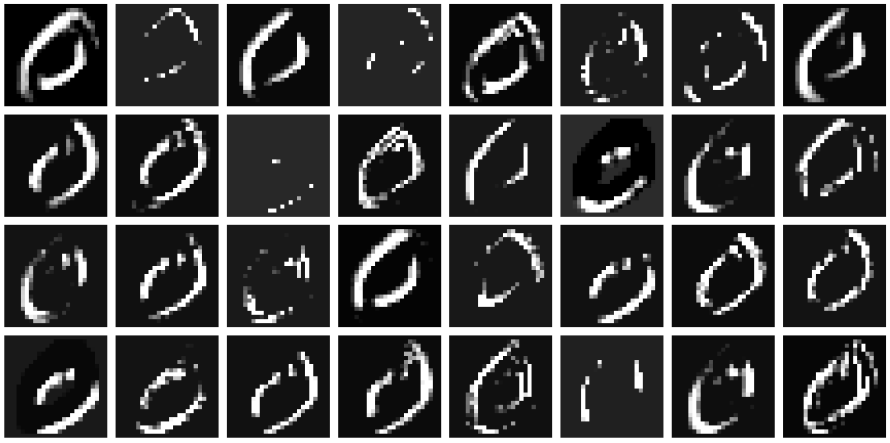
We also observe that almost all activation maps still retain almost all the information of the original image for both the 0 and 1. This is typical for CNNs, since activation maps extracted by layers become increasingly abstract, focusing on more detail the deeper you look into the CNN (Chollet 2017). Since our network only has one convolutional layer, we cannot show this effect. However, later on, when we use a deeper CNN for the classification of breast cancer, we will.

3.10 Kernels

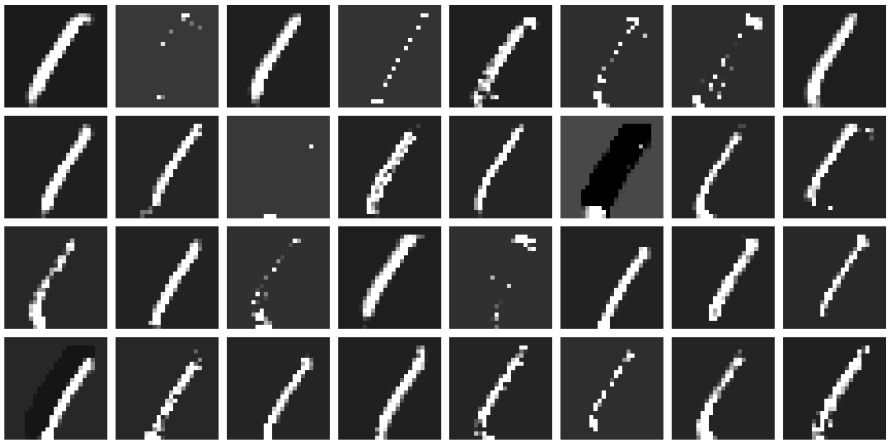
Another way to understand what is happening inside the CNN is to visualize the patterns that each kernel and its parameters is meant to respond to. This is achieved by gradient ascent in the input space, meaning we apply gradient descent to the values of an input image to maximize the response of a specific kernel. The resulting image is a representation of what the kernel is maximally responsive to (Chollet 2017).

To get the representations, we use a loss function designed to maximally amplify the activation as a result of a given kernel within the convolutional layer of the model, starting from a randomized input image. To adjust the randomized input image, stochastic gradient descent is used such that we iteratively achieve the maximum activation. To implement gradient descent, we compute the gradient of the loss with respect to the input of the model. The loss function used for the task is the mean value of the activation map. Since we are using stochastic gradient descent, we need to define a learning rate and a number of iterations for updating. In this study, a learning rate of 0.1 over 200 iterations is used. In Fig. 8, we show the results of each of the 32 kernels used in the model.

Observing the activation maps in the previous section, we can already create some first suspicions of what the model learns. Now, using the kernel visualizations,



(a)



(b)

Figure 7: 32 activation maps of the convolutional layer of the CNN in Fig. 5 for the first 0-image and first 1-image in Fig. 1. The activation maps show simple directional edges in the digits while maintaining almost all information of the initial image.

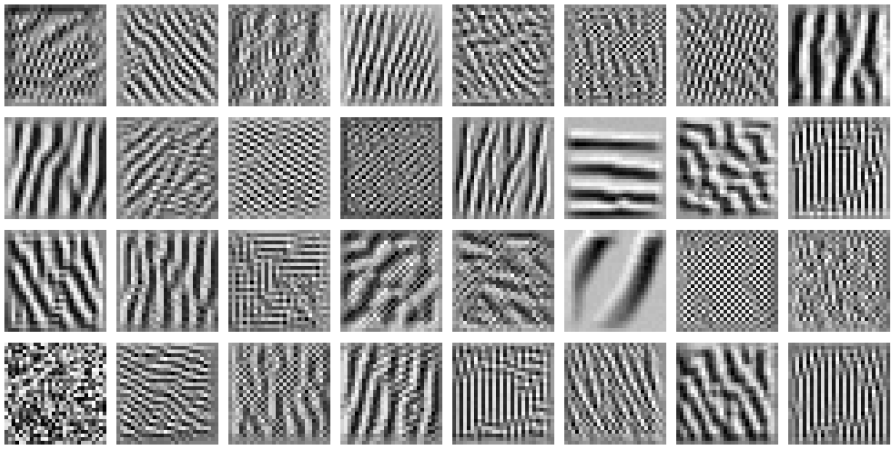


Figure 8: Visualizations of what patterns the kernels in the convolutional layer of the CNN model in Fig. 5 are most responsive to. The kernels are responsive to a combination of simple directional edges and circular contours.

those suspicions become clearer. In the illustrations, it is very clear to see how some kernels are very responsive to circular, horizontal, or vertical directions in the images. Similar to what can be seen in the previous section, since the network consists of very few layers, the kernels only encode simple directional patterns without a lot of details. However, this is sufficient when the images do not have much detail, such as the images that were used to train the network.

3.11 Heatmaps of Class Activation

The last visualization technique we use to understand the CNN model is called class activation map (CAM) visualization. It is used to visualize what regions within an input image are important for predictions from a CNN. It provides insight into which parts of the input image were most influential in leading to the model's decision. This can be very helpful when learning about the decision process of the model, especially in the case of misclassification.

A class activation heatmap represents a two-dimensional grid of scores computed for every location of the input image, indicating the significance of each location with respect to the class under consideration. Hence, CAM visualization allows you to generate a heatmap for the class 0, indicating how 0-like different parts of the image are, and also a heatmap for the class 1, indicating how 1-like parts of the image are.

In this study, to generate CAMs, we use Gradient-weighted Class Activation Mapping (Grad-CAM) proposed by Selvaraju et al. (2016). The method computes the gradients of the output of the last convolutional layer with respect to the activations, capturing how much each unit's activation contributes to the prediction. Then the gradients are averaged for each activation map in the last convolutional layer, resulting in a vector of pooled gradients. The vector highlights the importance of each activation map in the contribution to the prediction. Each activation map of

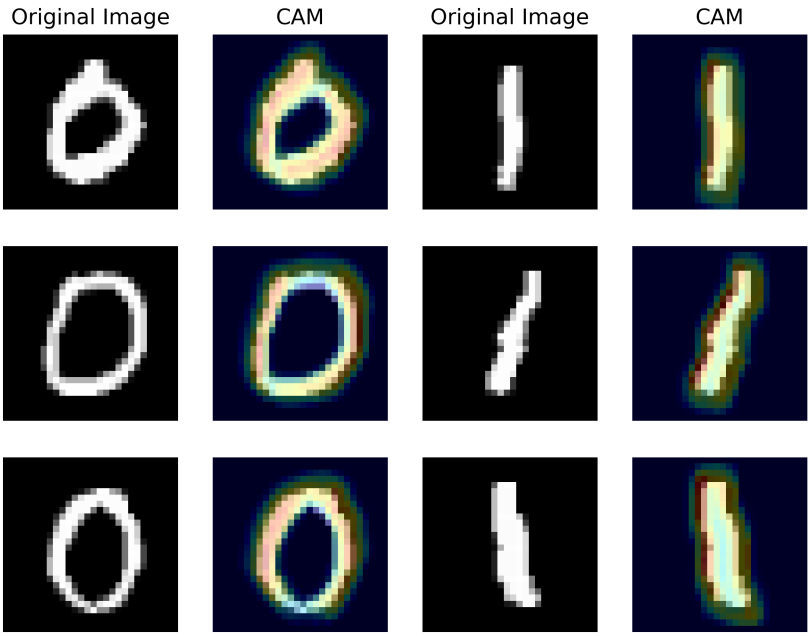


Figure 9: Class activation maps for a selection of images of the test set. The importance of a region for the classification decision of the model is indicated by the darkness of the color, with red being most important, yellow and green being less important, and blue even less important. The heatmaps perfectly align with the digits in the images, indicating the model correctly assigns the areas of the digits as important.

the last convolutional layer is then multiplied by the corresponding pooled gradient, weighing the activation maps based on their importance regarding the prediction. In the last step, 2-dimensional heatmaps are produced by averaging the weighted activation maps together. To visualize the CAM, the heatmaps are normalized and superimposed on the original image.

In Fig. 9, we show CAMs of a selection of 0-images and 1-images of the test set. From the heatmaps we observe that the model correctly identifies the distinctive regions of the 0-images and 1-images as most important, as the heatmaps show the most important regions for its classification decision align perfectly with the digits in the images. In particular, for the 0-images, the areas on the left and right side of the digits are most important, showing darker colors in those areas. For the 1-images, the most important region is the edge on the left side of the digit, with the heatmaps' darkest colors there.

3.12 Preliminary Conclusion

Now that we have used both the logistic regression and CNN model for the classification of 0's and 1's of the MNIST dataset, we can compare both and draw conclusions from it. Both models performed really well in the classification task

with the logistic regression and CNN model achieving an accuracy of 99% and 100% respectively. The logistic regression coefficients are visually well interpretable, with a clear indication what coefficients are important for a particular classification, and even the possibility to compute the magnitude of the effect of a certain pixel change on the probability that the image displays a 0 or 1.

On the other hand, although in a slightly different manner, the CNN model can be visually interpreted well too. The intermediate activations give a good indication on what is passed through the model as a result of what the kernels are responsive to. Additionally, the CAMs give useful information on the classification decision-making process, giving a solid intuition about the model.

Depending on the user's goals, one can choose either of the models. For instance, if the main goal is prediction accuracy, both models are adequate, but the logistic regression model is easier to implement, hence model preference probably resides with the logistic regression. On the other hand, if the user's goal is to interpret the model, again both can be adequate, but this time it additionally depends on how the user want its model to be interpretable. For logistic regression, additional statistical inference can be performed, while for the CNN this is not possible.

Now that we have covered both the logistic regression and CNN model, having used both for the MNIST classification task while trying to understand the models and compared their performance and interpretability, we conclude this section and continue with the actual classification task: predicting benign and malignant cases in breast mammography images. In the next section, we first discuss the CBIS-DDSM database used for the task and how it is preprocessed to suit the models, followed by how the logistic regression and the ResNet50 CNN architecture are used for the classification task.

4 Methodology

4.1 CBIS-DDSM

The dataset used in this study is an updated and standardized version of the Digital Database for Screening Mammography (DDSM) called the Curated Breast Imaging Subset of DDSM (CBIS-DDSM) published by Lee et al. (2017). The dataset was released to resolve the inability to replicate research in the field of CAD systems in mammography, due to the lack of a standard evaluation dataset. Although the DDSM database is useful, the CBIS-DDSM dataset was created to resolve issues encountered with the DDSM dataset. These issues involved unmaintained decompression code necessary to decompress and extract the images, incorrect ROI annotations, and a computationally very heavy task to process the images. Therefore, the CBIS-DDSM database was released to provide an easily accessible dataset with standard image formatting to resolve the replication issue, while simultaneously improve ROI segmentation with the help of trained radiologists and include patho-

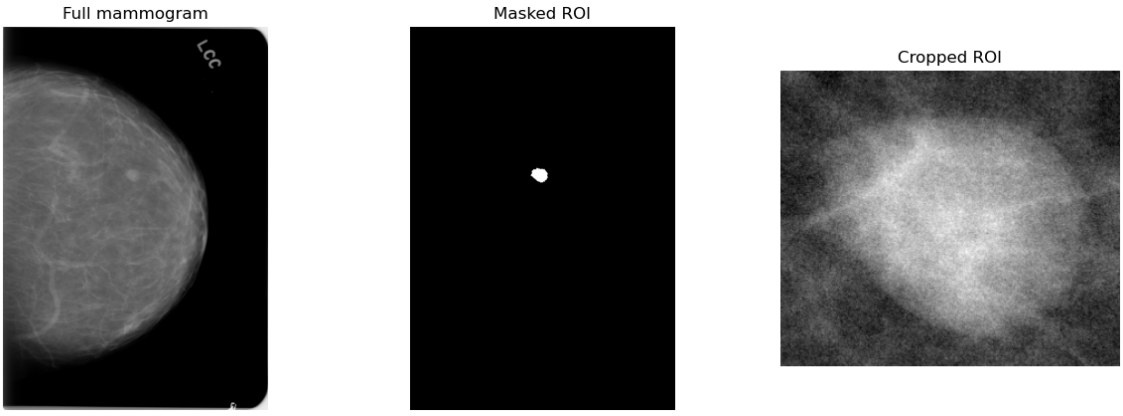


Figure 10: The three distinct images contained in the CBIS-DDSM dataset: a full mammogram (left), a masked ROI (middle), and a cropped convenience image of the ROI (right). In this study, the cropped convenience images are used.

logical diagnosis for training data.

The dataset comes with three types of images per breast: the original full mammogram, a masked ROI image, and a convenience image of the ROI zoomed in and cropped. In Fig. 10, we illustrate all three for one case.

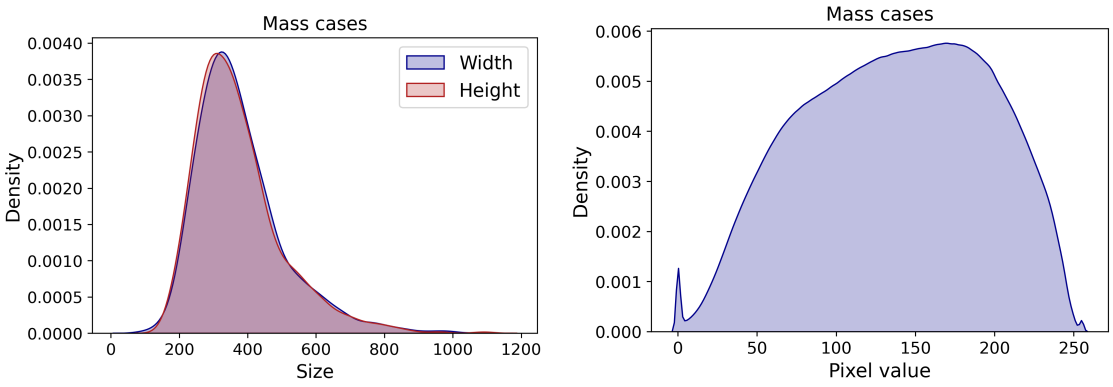
4.2 Exploratory Data Analysis

The dataset contains 891 mass cases, with most cases containing two versions, the standard mediolateral oblique (MLO) and cranio-caudal (CC) views. For MLO views, the image is taken from a lateral (side) angle to the medial (inner) side of the breast, while for the CC view, the image is taken from the cranial (top) to the caudal (bottom) of the breast. Due to the two different views, the dataset contains a total of 1696 images: 912 times benign and 784 times malignant. In this study, we will work with the cropped images and all data preprocessing steps and modeling will be carried out on only those. Consequently, from now on, references to images are references to the cropped images.

Typical image sizes are around 350 pixels and show Gaussian like distributions for width and height with fat tails on the right side, see left panel of Fig. 11. Since the ResNet50 architecture used in this study requires fixed-size input images, they will be resized in the data preprocessing steps discussed later in this chapter.

The pixel values of the images are distributed around 175, with a left-skewed characteristic, gradually declining towards a minimum of 0 and a maximum of 255, see right panel of Fig. 11. At 0, we observe a sharp peak caused by some of the outliers on the right side of the size distributions. In those cases, the images are not cropped enough to only show breast tissue and instead still show portions of the black background.

In Table 2, we show descriptives of both the image sizes and pixel values. On



(a) (b)

Figure 11: Sample distributions of the width and height of the images (a) and pixel values (b).

Metric	**Image size (W × H)**	**Pixel values**
Mean	377×372	137
Std	129×127	57
Min	95×159	0
Max	1086×1099	255

Table 2: Descriptive statistics of the image size and pixel values of the CBIS-DDSM dataset.

average, the images have size 377×372 and a pixel value of 137. The smallest image has size 95×159 and the largest size 1086×1099. The standard deviations of the image size and pixel values are 129×127 and 57 respectively.

In order to optimize the data for our model, we will perform data preprocessing steps that will help the model classify the instances. The three techniques that will be applied are image enhancement, image augmentation, and image resizing. Image enhancement is applied to increase contrast in the images and reduce noise, image augmentation is used to increase the dataset size, and the resizing of the images is done to comply with the required fixed-value image sizes necessary for our CNN. In the next section, we will discuss these techniques and why they were chosen.

4.3 Data Preprocessing

Before applying any of the data preprocessing steps, we first split the data into a training and test set. Of the 1696 images, we assign 80% to the training set and 20% to the test set randomly while maintaining the class balance.

4.3.1 Image Enhancement

Image enhancement is a key part in the field of image processing and is used to improve the quality of the images for their specific application. Generally, image enhancement's basic principle is to alter the information contribution of an image such that it suits the specific application better. Traditional image enhancement techniques are mostly focused on spatial and frequency domain processing, where the spatial domain techniques are based on directly processing the pixels in the images, such as the classic histogram methods, and the frequency domain techniques are based on converting the image to a certain frequency domain using a mathematical function such as Fourier Transform (Qi et al. 2021). However, newer methods have emerged, such as fuzzy theory, retinex models, and neural networks, each having their own advantages and disadvantages (Jobson et al. 1997; Zhou et al. 2007; Li et al. 2018). Due to the variety of advantages and disadvantages, it is hard to propose one universal method which suits the large variation of images present nowadays. Therefore, the image enhancement technique used should be chosen such that it suits the task at hand well.

In this study, the chosen image enhancement technique is a somewhat newer version of the traditional spatial domain method that uses Histogram Equalization (HE), at technique that has been widely used in the field of natural and medical image processing (Pizer et al. 1987; Pisano et al. 1998; Ragab et al. 2019). The advantage of spatial domain image enhancement is its simple understanding, relatively low complexity, and real time implementation. Moreover, with HE, no requirements of parameter settings of external factors are necessary, which makes it a straightforward and relatively easy method to implement. However, HE focuses on adjusting

the images globally, hence in some cases not being able to effectively improve local contrast, resulting in poor performance (Qi et al. 2021).

Therefore, Adaptive Histogram Equalization (AHE) was proposed by Zuiderveld (1994), which uses local histograms of predefined equal size grids of the image and redistributes the histograms to adjust the brightness and increase image contrast. However, one disadvantage of AHE is its unlimited possibility to redistribute the histograms, which can result in over enhancement of noise in the images (Ragab et al. 2019). To prevent this from happening, another method was proposed called Contrast Limited Adaptive Histogram Equalization (CLAHE), which uses a clip level to limit the amount of redistributing in the local histograms and the amount of contrast enhancement for each pixel (Sahakyan and Sarukhanyan 2012).

This study adapts CLAHE as its image enhancement technique. The algorithm can be summarized as follows:

1. The image is divided into equally size contextual regions.
2. Histogram equalization is applied to each region.
3. The histogram is limited by the clip level.
4. The clipped amount is redistributed among the histogram.
5. The enhanced pixel value is obtained using histogram integration.

In Fig (12), an example of an image, its CLAHE enhanced version, and both pixel value distributions are shown. The clip level and grid size used in this study are 2 and 8×8 pixels respectively.

4.3.2 Image Augmentation

CNN's have been proven to perform remarkably well on a wide range of computer vision tasks. However, the networks rely heavily on large datasets to avoid over-fitting (Halevy et al. 2009; Sun et al. 2017). Unfortunately, especially in medical areas, it is very difficult to create large datasets due to patient privacy, rarity of diseases, manual effort and expenses needed to carry out medical imaging processes, and the requirement of experts to label images. As a result, these obstacles have stimulated many studies to focus on image augmentation in the field of medical image classification (Perez and Wang 2017; Shorten and Khoshgoftaar 2019).

The goal of augmentation is to inflate the size of the training dataset by either oversampling or data warping. Oversampling uses techniques such as mixing images, generative adversarial networks (GANs), and feature space augmentations, to create synthetic image samples which are added to the training set. Data warping, on the other hand, does not create synthetic images, but instead modifies existing images by geometric and color transformations or random erasing (Shorten and Khoshgoftaar 2019).

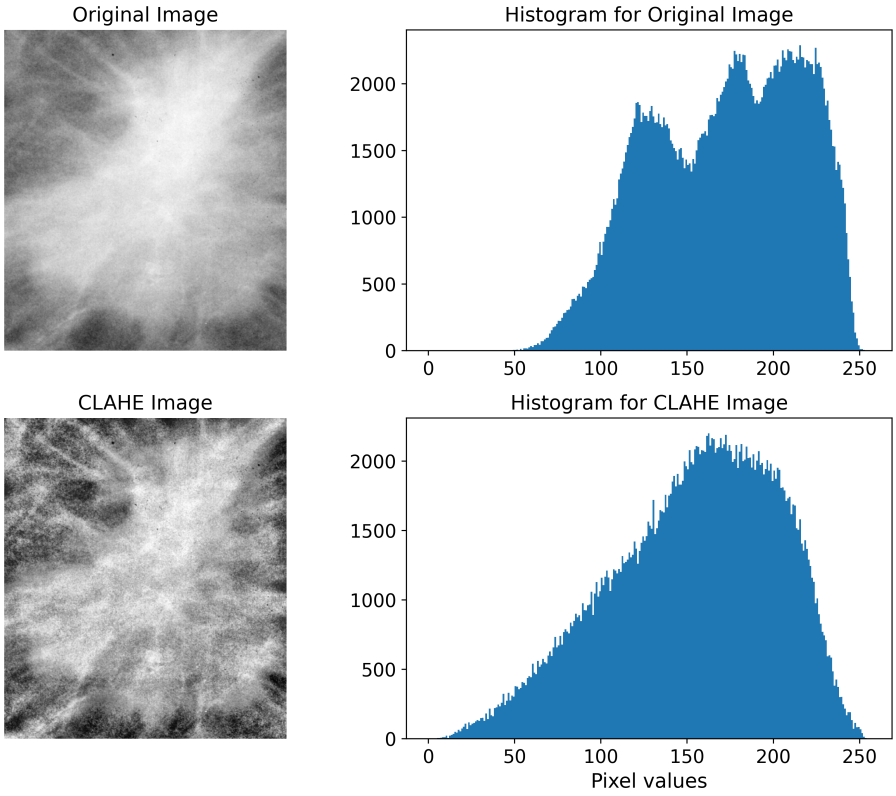


Figure 12: An example of an original mass lesions image (top left), its pixel value distribution (top right), its CLAHE enhanced version (bottom left), and the pixel value distribution of the enhanced version (bottom right). CLAHE is used to smooth pixel distributions and reduce noise.

Although there are many possibilities for image augmentation, there are not many comparative studies of the performance of the different possibilities. One study, carried out by Shijie et al. (2017), compared flipping, shifting, rotation, cropping, GANs, color jittering, PCA jittering, and adding noise. They found that rotation, GANs, cropping, and flipping were among the techniques that generally performed better than others. However, these techniques were not tested for medical image classification purposes. Therefore, in this study, the decision on the image augmentation technique is based on which technique the majority of the literature used for the CBIS-DDSM database, namely rotation (see Oza et al. (2022) for a review on image augmentation techniques for mammogram analysis). Each image in the training set is rotated by 0, 90, 180, and 270 degrees, resulting in a training set four times as large as the original set.

Additionally, although image augmentation is mostly focused on training datasets, numerous studies have shown the benefits of augmenting data of test sets as well (Shorten and Khoshgoftaar 2019). Using test set augmentation is similar to ensemble learning methods in the data space, as by augmenting both the training and test set, more robust predictions can be derived. Therefore, the same rotational augmentation is applied to the test set.

4.3.3 Image Resizing

In order to comply with the required fixed-size input image for our CNN, we need to resize the images. The ResNet50 architecture used in this study requires images to be 224×224×3, where the third dimension is usually used to identify colors (red, blue, and green). However, since our images are grayscale, we resize the image regardless of their size and then copy the information three times to get the desired 224×224×3 dimensions (hence the red, blue, and green color channels convey the same information). The 224×224×3 images are only used for the ResNet50 model, for the logistic regression we use the 224×224 images.

Traditional interpolation methods include bicubic, bilinear, or nearest neighbor interpolation. All three methods use neighboring pixels to create or merge pixels by taking the average of all neighbors. Bicubic takes 4×4 neighboring pixels, bilinear 2×2, and nearest neighbor uses only one neighboring pixel (Wang and Yuan 2014).

In this study, inter-area resampling interpolation is used, a similar method to the traditional approaches, except that the neighboring pixels over which is averaged, depends on the resizing scaling factor. For instance, if the image is being reduced by a factor of 3, then approximately 3×3 pixels in of the original image would contribute to a single pixel in the output image.

As mentioned before, 80% of the data is assigned to the training set and 20% to the test set. As a result, after all data preprocessing steps are applied, we have a total of 5424 training images and a total of 1360 test images. Of the test images, we assign 50% of the samples to a validation set, which is used during training to optimize the hyperparameters of the models. These hyperparameters will be

Set	Benign	Malignant	Total
Train	2916	2508	5424
Validation	364	316	680
Test	364	316	680

Table 3: Sample sizes and class balances for the training, validation, and test set used during modeling stages. The training set contains 80% of the data, while the validation and test set each contain 10%.

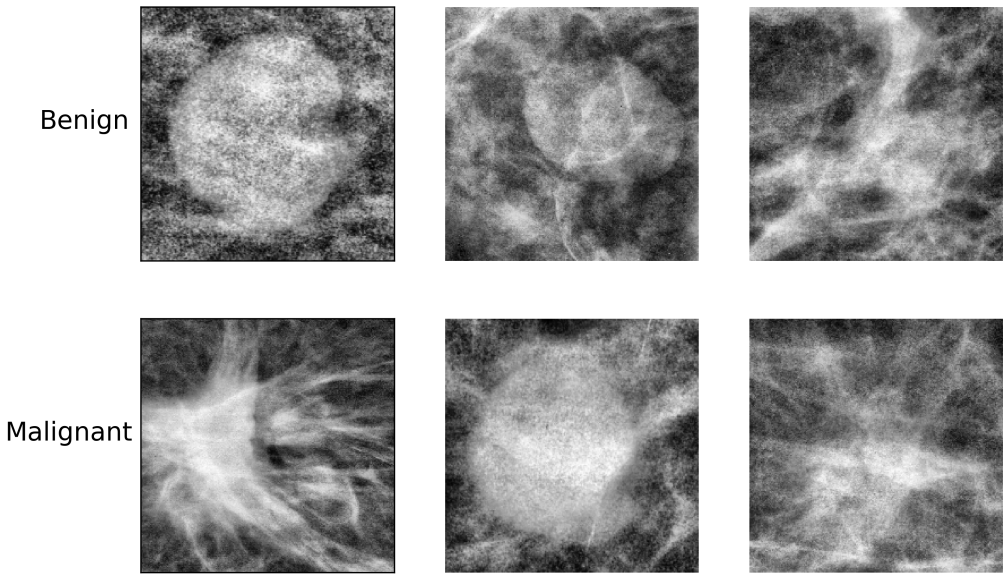


Figure 13: Collection of preprocessed samples used during modeling for both benign and malignant breast mass lesions.

discussed in the next section. In Table 3, we show a detailed overview of sample sizes and class balances per set. Lastly, in Fig. 13, we show a selection of the final preprocessed samples which are used during the modeling stage.

4.4 Modeling

In this section, we will cover additional theory about both the logistic regression and CNN model used for the breast mass lesion classification task and why this theory is necessary. Again, we will start with the logistic regression followed by the CNN model.

4.4.1 Regularized Logistic Regression

Typically, when MLE is used in combination with the logistic regression model, all the coefficient estimates will be nonzero. In the case that the amount of regressors d is very large, interpretation of the model can become very challenging. This is

especially true when dealing with 'Big Data', meaning the amount of regressors is much larger than the amount of observations ($d \gg N$). In fact, when dealing with $d \gg N$, the MLE estimates are non-unique, meaning there exist an infinite set of coefficients that maximize the objective function and which almost surely overfit the data (Hastie et al. 2015).

Since we are following the same logic for the logistic regression model as in the MNIST classification task, meaning one regressor for each pixel, we have a total of $224 \times 224 = 50176$ regressors, resulting in the following model:

$$\Pr(Y = 1|X_1, \ldots, X_{50176}) = \frac{\exp\left(\beta_0 + \sum_{j=1}^{50176} \beta_j X_j\right)}{1 + \exp\left(\beta_0 + \sum_{j=1}^{50176} \beta_j X_j\right)}. \tag{31}$$

Each of the coefficients has to be estimated using a much smaller total of 5424 observations. Therefore, to overcome the interpretation and convex issues, we need to constrain or regularize the estimation process.

In this study, the lasso or l_1-regularization is used. Given the logistic regression model in Eq. (31), the log-likelihood function with lasso-regularization takes the form

$$\log f(\beta_0, \beta) = \frac{1}{n} \sum_{i=1}^{n} \Big(Y_i \log(P_i) + (1 - Y_i) \log(1 - P_i) \Big) + \lambda \|\beta\|_1, \tag{32}$$

where P_i is Eq. (31) for a given $X_i = (X_{i,1}, \ldots, X_{i,50176})$, $\|\beta\|_1 = \sum_{j=1}^{50176} |\beta_j|$, and λ is a user specified hyperparameter indicating the regularization strength. Higher values for λ correspond to a more heavily-constrained model, and vice versa.

The reason why the l_1 norm is often chosen above other regularizations such as l_2 or any other l_q norm is that for a large enough λ, the lasso solution is a sparse solution, meaning only a relatively small number of coefficients play an important role, and hence only some coefficients are nonzero. In this way, it automatically provides a way of doing model selection in logistic regression while increasing the model's interpretability.

However, careful consideration of λ is necessary when using the l_1 norm, since it controls the complexity of the model. Lower values of λ allows for better adaptation to the training data, while higher values allow for better interpretability. Hence, modeling λ is often a trade-off between the performance and interpretability of the model. A common approach used for determining λ is through cross-validation. This method, however, aims to find the optimal λ by evaluating the performance, thereby neglecting interpretability (Hastie et al. 2015).

The objective in Eq. (32) is convex and the likelihood part is differentiable. Therefore, the MLE can be computed again using the same numerical methods mentioned in Section 3.2. For this part of the study, the same algorithm is used as in the preliminary phase: SAGA. After estimation, we again consider the pseudo-R^2 and accuracy of the model, as well as two additional evaluation metrics. These metrics will be discussed later in Section 5.2. Contrary to the preliminary phase, however, we do not compute the LRT-statistic and p-value since the statistic is

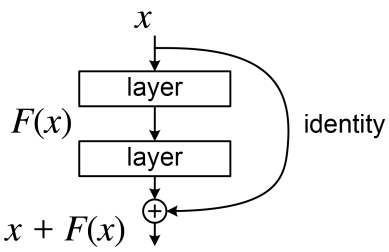


Figure 14: Skip connection used in residual networks. The output x of a layer is directly added to some layer further down the network. This helps avoid the vanishing gradient problem and prevents the large amount of layers to be the reason for bad performance.

not χ^2-distributed anymore. Distributional results for lasso exist but are hard to establish and are beyond the scope of this study. Earlier work on this can be found in Knight and Fu (2000). For more recent work, see van de Geer et al. (2013) or Zhang and Zhang (2011).

Also, contrary to the preliminary phase, after having covered everything necessary to apply the model to the data, we do not continue with showing the results right away. Instead, we first cover extra material on the CNN used for the CBIS-DDSM dataset, and then discuss all results at once to follow the more traditional set up.

4.4.2 Residual Networks

A treat often observed when training very deep neural networks is that with increasing network depth, the performance saturates and even degrades, even though overfitting is not the cause (Baccouche et al. 2022). To overcome this issue, ResNets or residual networks were introduced by He et al. (2015), that use residual blocks with skip connections. These residual blocks, consisting of sequences of layers, are designed such that they do not necessarily have to learn the mapping from inputs to outputs, but can learn a residual function of the difference between input and output. This is achieved by adding skip connections that bypass the block's layers, see Fig. 14.

As shown in the figure, the output x of a layer is directly added to the output of another layer $F(x)$ further down the network. For instance, given an input passed through a ReLU activation, instead of passing the result directly into the next set of layers, it is also added to the output of a layer further down the network after another ReLU activation.

These skip connections help avoid the vanishing gradient problem discussed in Section 3.7 by allowing the gradients to flow through the network more effectively during training. Additionally, it ensures that the large amount of layers in the network do not harm its performance, since these layers can simply learn to approx-

imate an identity function if that is most beneficial for reducing the training error (He et al. 2015).

4.4.3 ResNet50 Base Model: Transfer Learning and Fine-Tuning

One of the most well-known ResNet architectures is ResNet50. Originally, the network was designed for the prestigious ImageNet Large Scale Visual Recognition Challenge (ILSVRC) competition, where researchers took on the challenge to classify 1,000 categories from a dataset containing 14 million images. In 2015, the architecture won the competition, and since then it has been proven to succeed in other areas such as medical imaging applications (Szegedy et al. 2016; Garcia-Gasulla et al. 2017; Baccouche et al. 2022).

The network consists of 50 layers divided over 5 consecutively placed residual blocks and a fully connected layer for the classification of 1,000 classes. Each residual block consists of a convolutional and identity block with sequences of convolutional layers, ReLU activation layers, and skip connections. In Fig. 15, we illustrate the model in detail, with the full ResNet50 model in the left top corner, a residual block in the right top corner, and a convolutional and identity block with skip connections in the left and right bottom corner respectively.

Training deep learning models such as the ResNet50 often requires large amounts of data to optimize the large number of parameters. However, as mentioned before, due to patient privacy, rarity of diseases, manual effort and expenses needed to carry out medical imaging processes, and the requirement of experts to label images, most medical imaging datasets that are available suffer from a small number of observations. Apart from data augmentation, a method called transfer learning can be applied to help overcome this problem. Transfer learning is a common solution used in many medical imaging applications (He et al. 2016; Ragab et al. 2019) by pre-training a model on a very large and diverse dataset. As a result, the model's earlier layers will learn universal patterns like boundaries, curves, and edges that are in general important for image classification. After that, the pre-trained model is retrained using the dataset of interest for the final classification task. This method provides faster and generalizable training for deep learning models with small datasets (Baccouche et al. 2022).

It should be mentioned, however, that often these large datasets do not have the same amount of classes. For instance, the ImageNet dataset has 1,000 classes compared to the 2 classes of our dataset. Therefore, after pre-training the network, it has to be fine-tuned by removing the last classification layer and add one or multiple layers that comply with the classification task of the specific dataset.

In this study, we use transfer learning in combination with fine-tuning to the base ResNet50 model. The network is first pre-trained using the ImageNet dataset, and then fine-tuned by replacing the last classification layer for a set of new layers. More specifically, we remove the last classification layer and flatten the output of the layer before, followed by a dropout layer and the classification layer for the benign and

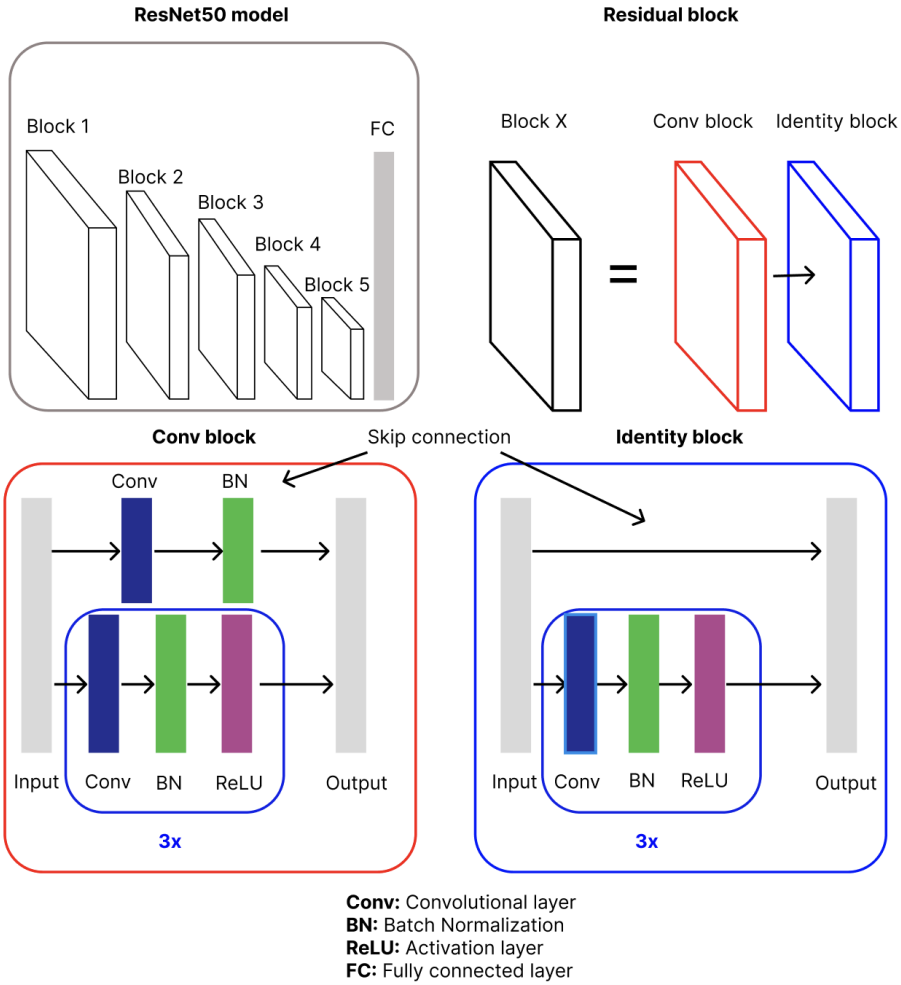


Figure 15: An overview of the full ResNet50 model, containing five residual blocks and a fully connected layer for classification (top left). A residual block, consisting out of a convolutional block and identity block (top right). The convolutional block in detail, containing a skip connection and 3 sequences of convolutional, batch normalization, and ReLU activation layers (bottom left). The identity block in detail, containing a stricter skip connection but the same sequence as the convolutional block (bottom right).

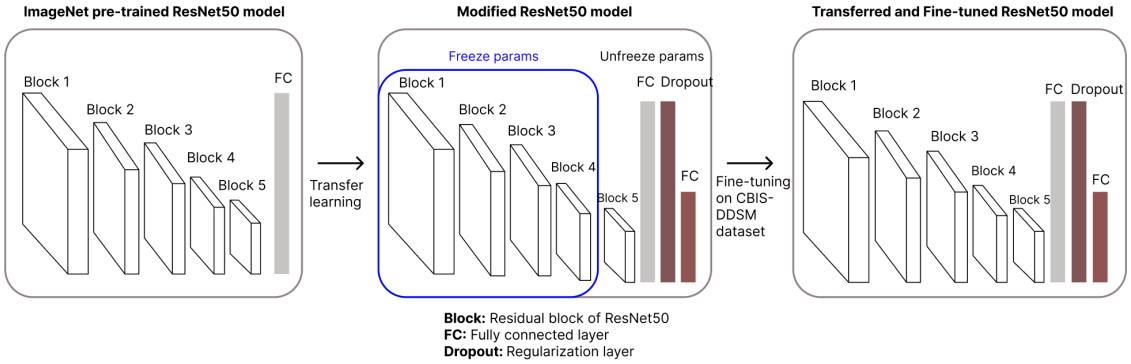


Figure 16: Visual representation of the full methodology employed in this study for the ResNet50 model. First, the model is pre-trained using the ImageNet dataset for the classification of 1,000 classes (left). Then, the top layer of the pre-trained model is fine-tuned by replacing it for a sequence of fully connected layers with dropout, and the parameters of the first four residual blocks are frozen (middle). Finally, the model is re-trained on the CBIS-DDSM dataset.

malignant classes. Additionally, while fine-tuning, we freeze the first four residual blocks, meaning during training on the CBIS-DDSM dataset, only the parameters of residual block five and the following layers are trainable. This is done to make sure the general learned patterns in earlier layers are not adjusted, and the dataset specific patterns learned in later layers are retrained. In Fig. 16, we illustrate the full training process used for the classification task.

5 Results

5.1 Data Preprocessing

Mammograms are typically acquired through a digital X-ray mammography scanning machine, which compresses the breast and, as a result, can degrade the image quality. Therefore, data preprocessing steps have been applied to remove noise and adjust the data using a histogram equalization technique that smooths pixel distributions. Additionally, since neural networks rely heavily on the dataset size to overcome overfitting and generalize well, data augmentation was used. Consequently, every image was augmented to four using rotational augmentation with degrees 0, 90, 180, and 270. Furthermore, since the ResNet50 model requires an input image of size 224×224×3, we resize the images to 224×224 using inter-area resampling interpolation and copy the 2-dimensional image three times to get 224×224×3. For the logistic regression only the 224×224 samples were used. Then, in the last data preprocessing step, for the ResNet50 model, the input was normalized to a range of [0, 1]. Samples of input images used for the classification task are shown in Fig. 13.

5.2 Evaluation Metrics

In the preliminary phase of this study, we focused only on the accuracy of both models, a metric that gives a good performance indication when the classes are balanced. Although the benign and malignant classes are well-balanced in the CBIS-DDSM dataset as well, allowing us to use the accuracy again, medical classification tasks ask for more insights. For instance, when it is crucial not to miss a malignant case (for obvious health reasons), but also crucial not to identify a benign as a malignant case (for unnecessary patient stress, health insurance costs, and health implication due to treatment), we can use the sensitivity and specificity scores. The sensitivity measures the proportion of actual positives that are correctly identified by the model, while the specificity measures the proportion of actual negatives that are correctly identified. They are expressed as follows:

$$\text{Sensitivity} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}} \tag{33}$$

$$\text{Specificity} = \frac{\text{True Negatives}}{\text{True Negatives} + \text{False Positives}}. \tag{34}$$

In our case, positives refer to the malignant class and negatives to the benign class. Both scores range between 0 and 1, with 1 being a perfect score.

Besides the accuracy, sensitivity, and specificity scores to evaluate the models and compare performance, we also compute the pseudo-R^2 and the sparsity percentage of the logistic regression. The former being a measure of improvement of the fit compared to a null model without regressors, and the latter being the percentage of coefficients estimated to be zero.

5.3 Hyperparameter Tuning

The full dataset was randomly split into groups of 80% for training, 10% for validation, and 10% for testing. In Table 3, we show sample sizes and class balances for each set. For both models, we used cross-validation to compute the optimal hyperparameters by training the models on the training set and evaluating performance on the validation set using the accuracy. The logistic regression used was lasso regularized, resulting in only one user specified hyperparameter λ, indicating the strength of regularization. To estimate the model, maximum likelihood estimation was used, which was solved using the numerical method SAGA.

For the ResNet50 model, we cross-validated over the batch size, patience, learning rate, dropout rate, and the l_2-regularization strength. Due to limitations in the hardware used in this study, only one cross-validation fold was used for both models. In other words, each combination of parameters is optimized and evaluated just once on the full training and validation set. The model was optimized using the categorical crossentropy loss function and Adam optimization.

In Table 4, we show the range of parameter values that were explored and the ones that were optimal (in bold). Conclusively, the best evaluation for the logistic

Model	Parameter	Values explored	Description
Logistic	λ	0.01, 0.2, **0.5**, 2, 4, 6, 10	l_1 regularization strength
ResNet50	Batch size	20, **32**	Mini-batch training size
	Patience	**5**, 10	Improvement patience
	Learning rate	$\mathbf{10^{-5}}, 10^{-4}, 10^{-3}$	Step size of Adam optimizer
	Dropout	40%, 50%, **60%**	% of units dropped in layer
	α	$10^{-1}, \mathbf{5 \cdot 10^{-2}}, 10^{-3}, 5 \cdot 10^{-4}$	l_2 regularization strength

Table 4: Overview of each of the hyperparameters used in this study for both the logistic regression and ResNet50 model. We explored a range of values per hyperparameter and computed the optimal hyperparameters (here in bold) using 1-fold cross-validation.

Model	Accuracy	Sensitivity	Specificity	Pseudo-R^2	Sparsity %
Logistic	0.61	0.54	0.65	0.87	25.55%
ResNet50	0.79	0.73	0.84	-	-

Table 5: Breast mass lesion classification results for both the logistic regression and ResNet50 model.

regression was reported for a λ of 0.5, while for the ResNet50 it was a batch size of 32, a patience of 5, a learning rate of 10^{-5}, a dropout rate of 60%, and an α of $5 \cdot 10^{-2}$.

5.4 Quantitative Classification Results

The pathology classification results are compared between the logistic regression model and the ResNet50 model using the accuracy, sensitivity, and specificity. Additionally, we evaluate the logistic regression model on the pseudo-R^2 and the sparsity percentage. The result are shown in Table 5.

The comparative results show that the proposed ResNet50 model outperformed the logistic regression for all three metrics. The logistic regression achieved an accuracy, sensitivity, and specificity of 0.61, 0.54, and 0.65 respectively, while the ResNet50 model achieved scores of 0.79, 0.73, and 0.84 respectively. Hence, for all three metrics, the ResNet50 model outperformed the logistic regression by approximately 20%. Consequently, the results emphasize the advantage of the ResNet50 model in the classification task compared to the logistic regression.

Additionally, for both models, we observe the lowest scores for sensitivity, indicating that they both relatively have more difficulty identifying actual malignant cases. Contrary, we observe the highest scores for specificity, indicating that both models relatively find it easier to identify actual benign cases.

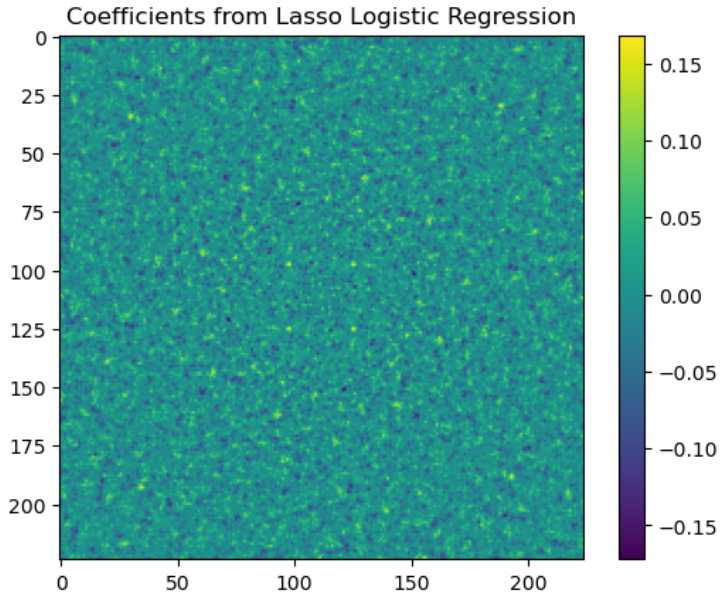


Figure 17: Heatmap visualization of the coefficients of the logistic regression model in Eq. (31) used for the classification of the breast mass lesions images. No visually interpretable patterns can be found. In fact, the coefficients show noise-like characteristics, with alternating areas of positive and negative coefficients.

Finally, for the logistic regression model, we found a pseudo-R^2 and sparsity percentage of 0.87 and 25.55 respectively. The former indicates that the model has an improved measure of fit compared to the null model without regressors, while the latter tells us that 25.55% of the 50176 coefficients were put to zero by the lasso regularization.

5.5 Visualization Results

Apart from evaluating both models using evaluation metrics, we also try to interpret the models using visualizations. For the logistic regression, this means visualizing the estimated coefficients as a 224×224 heatmap and interpreting this heatmap using the logic discussed in Section 3.2.1. On the other hand, for the ResNet50 model, we try to interpret it by showing intermediate activation maps, intermediate kernel representations, and class activation maps. We start with the logistic regression, followed by the ResNet50 model.

5.5.1 Logistic Regression Coefficients

After estimating the 50176 coefficients, we transform our 50176-dimensional coefficient vector to a 224×224 matrix and show this matrix as a heatmap. The results are shown in Fig. 17. Unlike the coefficients of the logistic regression used for the MNIST classification task, the coefficients for this task do not show any recogniz-

able patterns. In fact, the heatmap shows noise-like characteristics with areas of positive coefficients (yellow color) and areas with negative coefficients (dark blue color), making it visually uninterpretable.

The reason for this can be imposed on the characteristics of the CBIS-DDSM data itself. While the 0-images and 1-images of the MNIST dataset clearly have distinctive areas of importance for the digits, and a clear difference in shape, the mass lesions of the CBIS-DDSM dataset do not. For instance, for the 0-images and 1-images (see Fig. 1), high valued pixels right in the middle of the image contain a lot of information about the image displaying the digit 1, marking this area as important for a 1-classification, and thus being able to observe that in the coefficient heatmap. Such reasoning cannot be used with the breast lesion images. Both malignant and benign cases can have very similar shape, with the most important areas being in the same location of the image.

Moreover, from the average 0-image and 1-image (see Fig. 2), we see that the digits are very consistently placed at the same location in the image, contrary to the images of the breast lesions. Therefore, the coefficients cannot clearly assign certain pixel areas to a particular class, resulting in alternating areas of positive and negative coefficients.

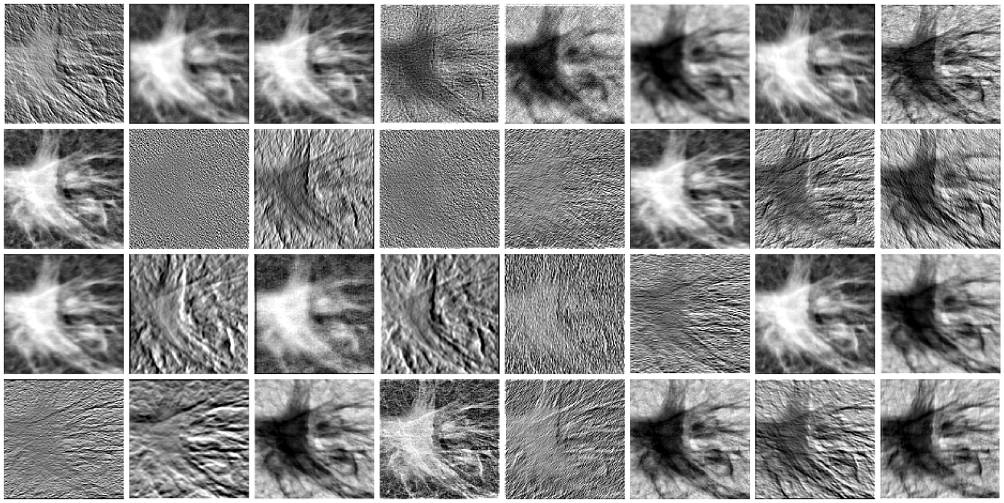
5.5.2 Intermediate Activations

To illustrate the visual representations that are passed through the ResNet50 model, we show intermediate activation maps produced by the convolutional layers for the first malignant sample shown in Fig. 13. We do so by showing the first 32 activation maps of the first convolutional layer of each residual block of the ResNet50 model. Hence, we get five illustrations of collections of activation maps for each of the five residual blocks. The results are shown in Figs. (18a) through (18e).

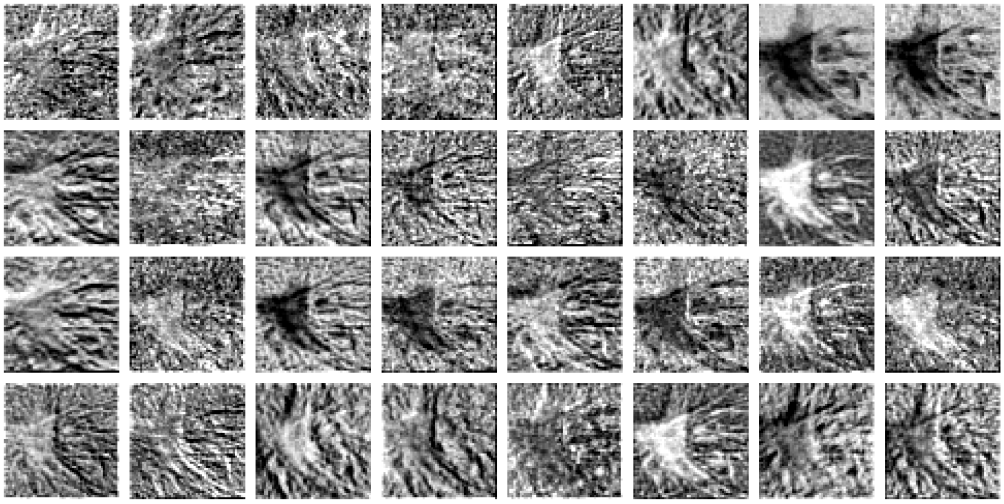
The collection of activation maps of residual block 1 shows similar features as the first convolutional layer of the CNN used for the MNIST dataset. That is, a collection of some edge detectors where almost all information of the input image is retained. In some cases, the black and white areas of the initial image are switched to white and black, indicating the model is gathering color information.

For block 2, the layer is increasingly focused on edges and small details of the image, making the maps more abstract. However, in almost all cases, the malignant lesion can still be seen and the maps are visually still interpretable. This interpretability is partly lost for the first time in block 3. Although in some cases, the initial lesions can still be identified, in most cases, the information from the maps is too abstract to visually interpret.

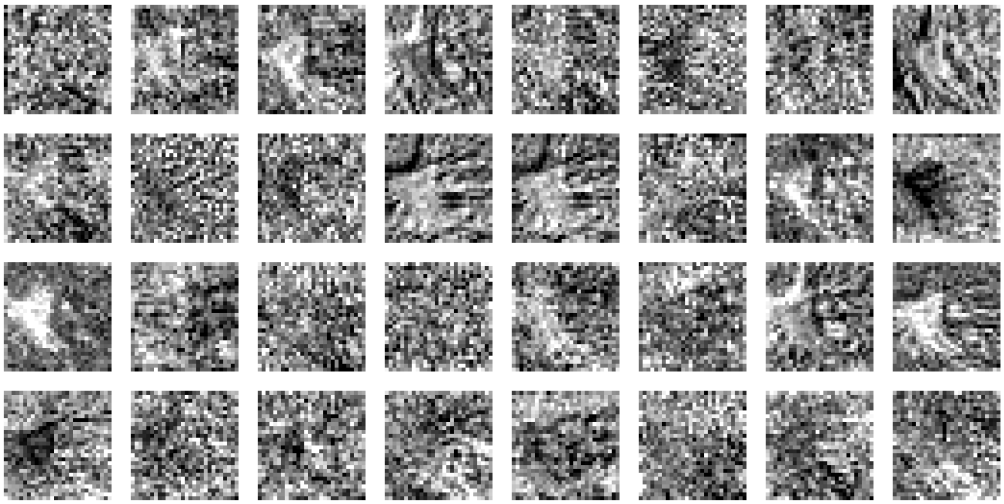
This trend is continued in the deeper blocks of the network. In block 4 and 5, none of the maps are visually interpretable, and the size of the maps become smaller and smaller until they only have size 7×7 in the last block. The observed trend is often described as a universal characteristic of deep neural networks. The



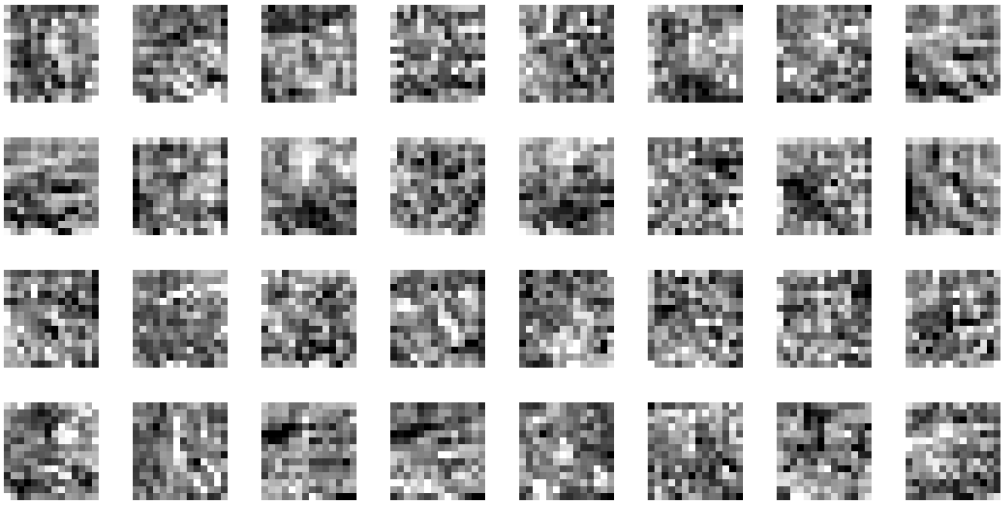
(a) Residual block 1.



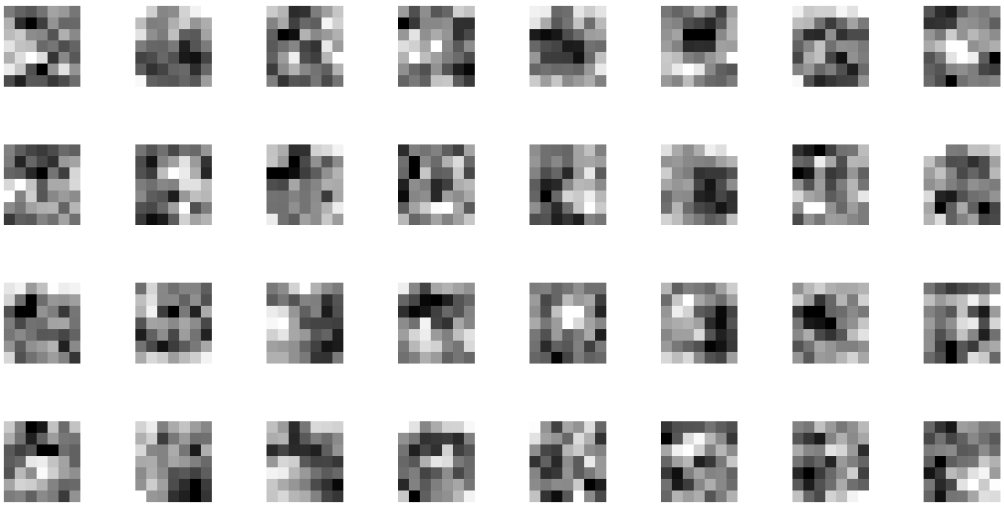
(b) Residual block 2.



(c) Residual block 3.



(d) Residual block 4.



(e) Residual block 5.

Figure 18: Activation maps of the first convolutional layer in residual block 1 through 5 (panel (a) through (e) respectively) for the first malignant case in Fig. 13. In the first two blocks, the activation maps contain almost all the information from the initial image and we see focus on simple directional edges. Deeper into the layers, the activation maps become increasingly small and abstract, and visual interpretation is impossible

activations carry a decreasing amount of information about the input image, but an increasing amount of information about what class it is (Chollet 2017). This is the result of units in deeper activation maps being nonlinear combinations of information in the image.

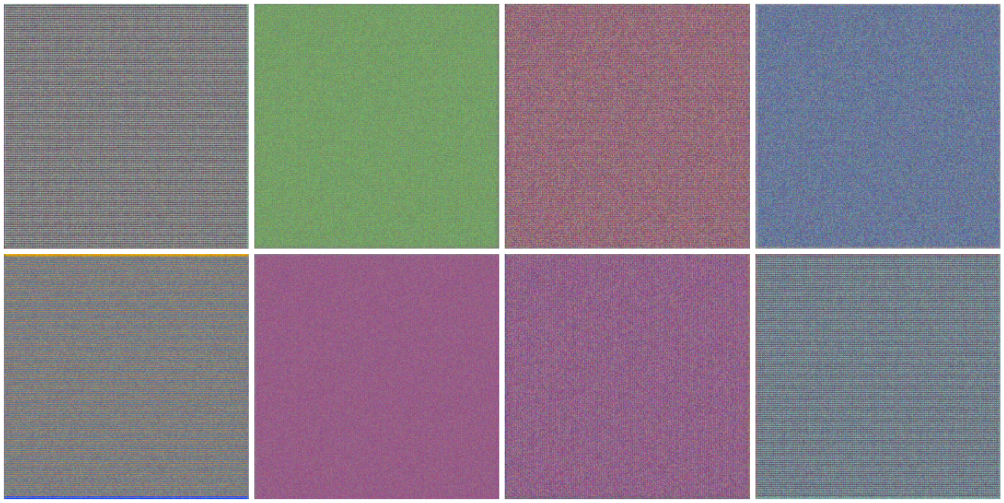
5.5.3 Kernels

Next, we show what the kernels in the same layers as for the preceding activation maps, are maximally responsive to. In other words, we show what patterns the model is looking for in each of the layers using the method discussed in Section 3.10. Similarly to this section, a learning rate of 0.1 over 200 iterations is used to create the representations. The results are shown in Fig. 19.

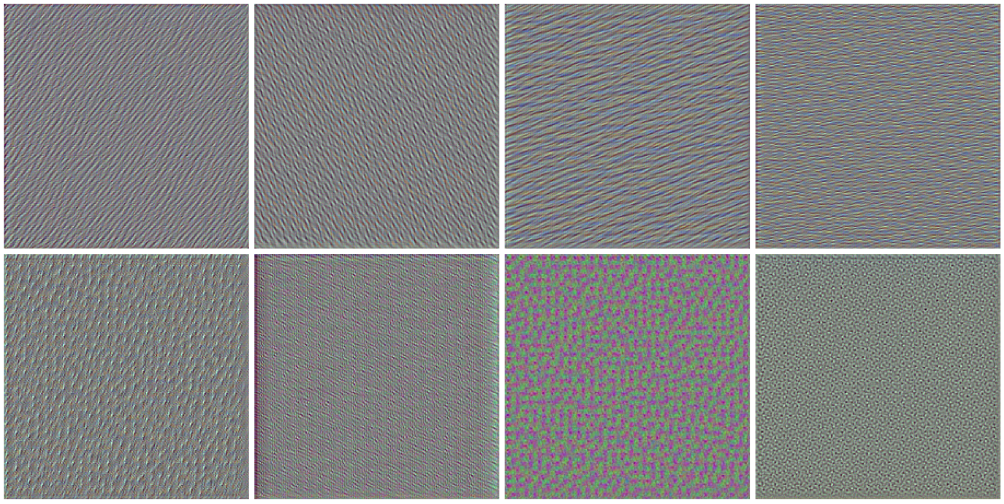
In the first block, we observe that the kernels are not yet responsive to any particular patterns, but instead focus on colors of the input. This is a result of the ResNet50 model itself, requiring a third (color) dimension of the input images (224×224×3). Since our images contain the same information for each of the color channels, no useful information will be gained from this. This changes for the kernels in block 2 of the model. Here we observe the kernels start to encode simple edges in varying directions, with one kernel being responsive to dotted patterns.

This trend continues into block 3, where we observe that the patterns the kernels are looking for start to resemble increasingly specific textures. Those textures become even more apparent in the representations of block 4 and 5, where we observe very distinctive patterns. Some showing large circular contours while others displaying small and repetitively patterns. It is clear, however, that the deeper into the model, the more the models looks for specific patterns.

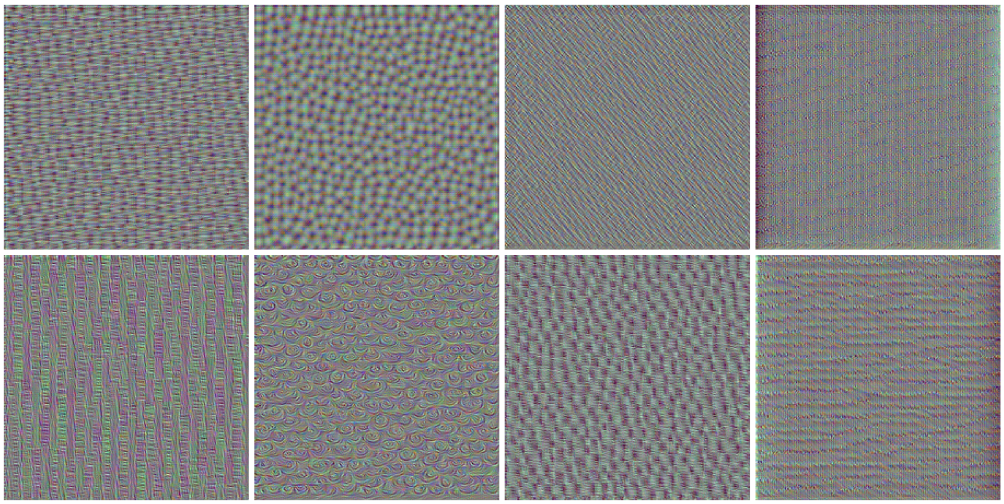
These patterns are obviously not visually present anymore in the activation maps of the previous section. However, in the context of the entire network, those patterns might be constructed from a series of transformations starting from the original input image, through all the layers leading up to the one we are visualizing here.



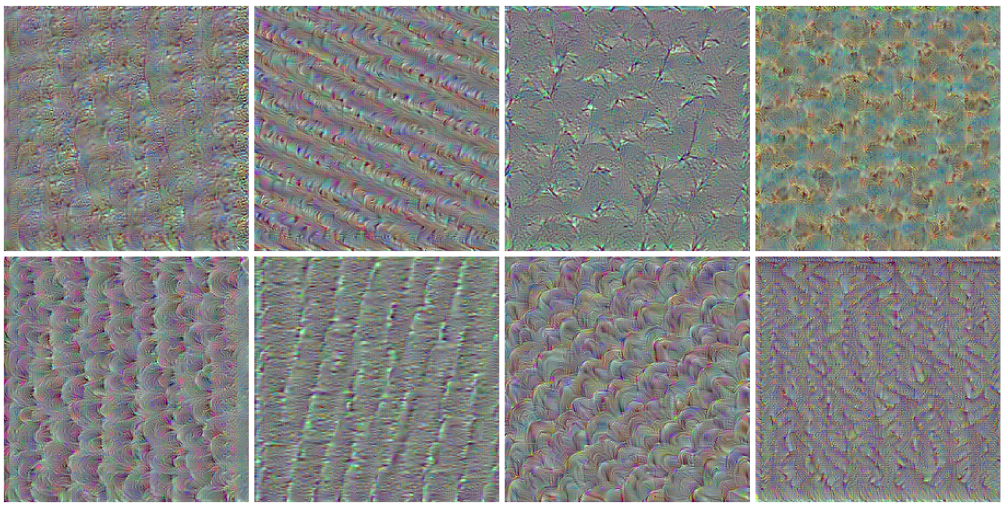
(a) Residual block 1.



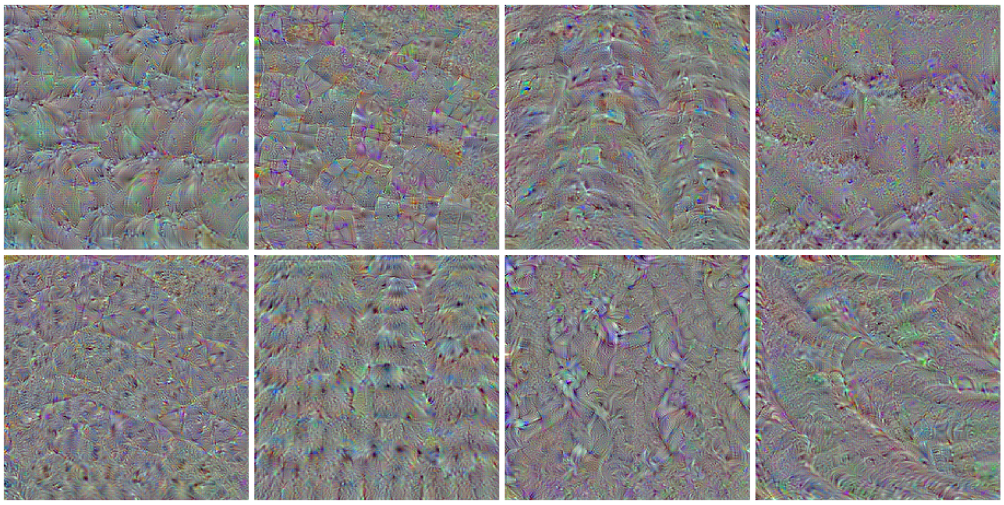
(b) Residual block 2.



(c) Residual block 3.



(d) Residual block 4.



(e) Residual block 5.

Figure 19: Visualization of what kernels of the first convolutional layer of residual block 1 through 5 (panels (a) through (e) respectively) are most responsive to. In earlier blocks, the kernels are looking for colors or simple directional edges and patterns. The deeper into the network, the more specific those edges and patterns become.

5.5.4 Heatmaps of Class Activation

In the last visualization technique, we show heatmaps of class activations, which show what areas of the input image were most important for the model in making its classification decision. The heatmaps were computed using a method called Grad-CAM, discussed in Section 3.11. We will show correctly and incorrectly classified images for both benign and malignant lesions, with the original image next to it.

However, contrary to Section 3.11, where it is trivial to identify what parts of

the image are important for the class in the image, with breast mass lesion images, specific domain knowledge is required to separate benign from malignant. Therefore, due to the lack of such knowledge, we only discuss here the differences seen between the heatmaps of correctly and incorrectly classified images, and do not go into detail about the correctness or incorrectness of those heatmaps. The results are shown in Fig. 20.

In the figure, we show four incorrectly classified images and four correctly classified images with benign and malignant cases. The incorrectly classified images are displayed in the first and second columns, while the correctly classified images are shown in the third and fourth columns. Then the benign cases can be found in the first and second rows and the malignant cases in the third and fourth row.

From the figure, we observe that the model for the incorrectly classified images consistently assigns the outer parts of the images as most important for both the benign and malignant cases, leaving out the regions where the image has its most intense pixels. Contrary, for the correctly classified images, we observe that the model does identify the most intense regions as most important for its classification decision, highlighting it betters identifies the mass lesions in the image and therefore is able to make a better decision.

6 Discussion

In the previous section, we presented the comparative classification and visualization results for both models. In this section, we discuss our methodology, starting from the data preprocessing until the modeling for parts where we think considerations can be made for future research.

6.1 Data Preprocessing

In this study, we conducted the final stage of the traditional CAD system: diagnosing and classifying breast mass lesions. We conducted this stage on convenience cropped images of the CBIS-DDSM dataset. To enhance the images and reduce noise, we used the CLAHE enhancement technique. This technique requires the user to specify two parameters: the clip level and the grid size. Although CLAHE is a widely used enhancement technique for medical images (see e.g. (Al-antari Aisslab et al. 2018; Ragab et al. 2019; Baccouche et al. 2022)), the user-specified parameters are rarely reported. Therefore, the choice of this study's clip level and grid size was based on a correspondence with the author of one of the most successful CBIS-DDSM classification papers discussed in the Section 2) (Baccouche et al. 2022). To test these parameters, we trained and evaluated the models on non CLAHE enhanced images and CLAHE enhanced images. The results showed improvement for the CLAHE enhanced images. However, to be more robust to the effects of CLAHE, the parameters could be taken into the cross-validation process to find the optimal

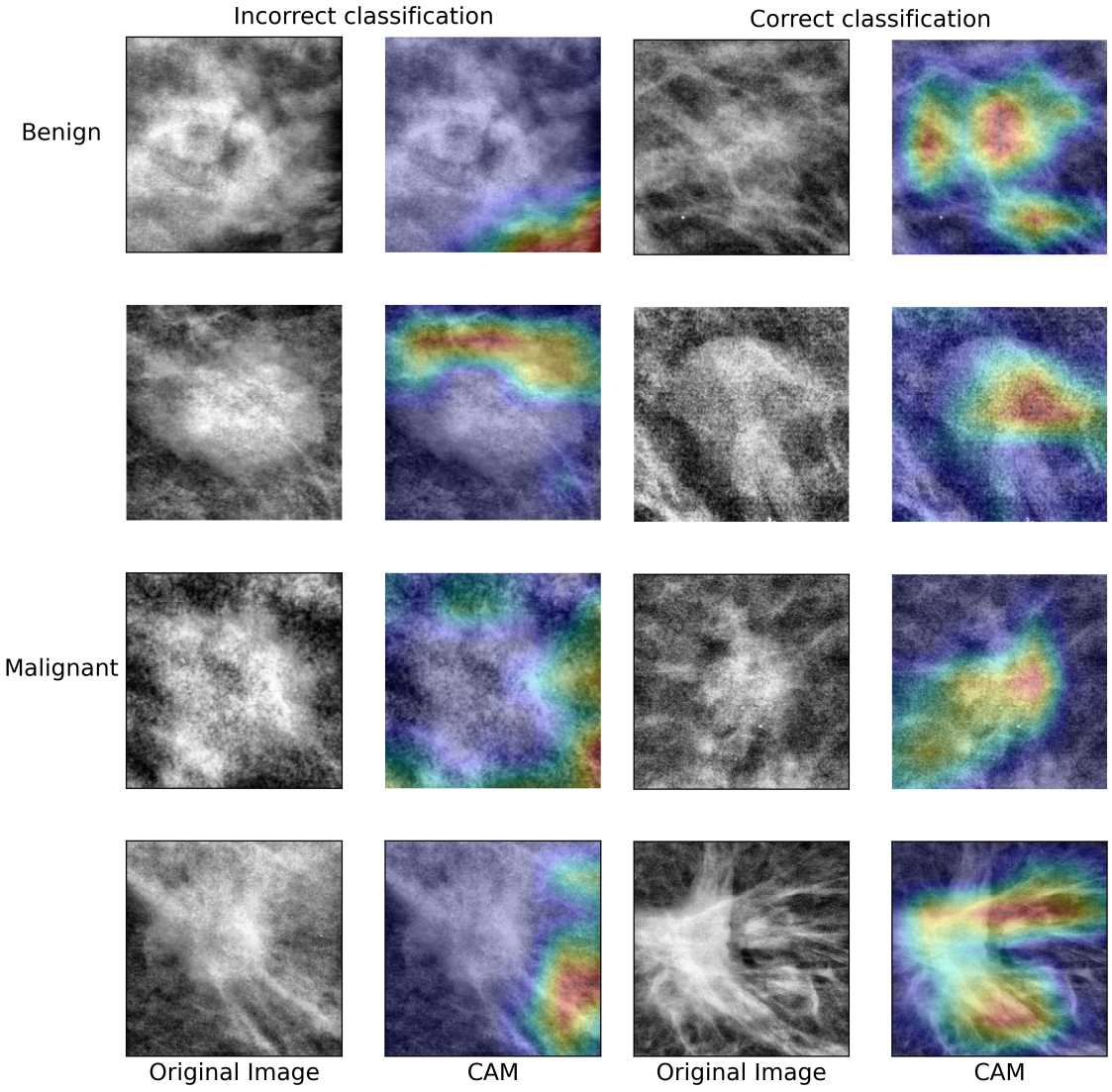


Figure 20: Original breast mass lesion images and their corresponding class activation maps for incorrectly classified images (first and second columns) and correctly classified images (third and fourth columns) for benign cases (first and second row) and malignant cases (third and fourth row). The model fails to identify the high intensity regions of the image for the incorrectly classified images and assigns outer edges of the image as most important. Contrary, for the correctly classified images, it identifies the high intensity regions well, assigning those as important for classification.

values.

Another point of consideration is the resizing of the images. Since the ResNet50 model requires input images to be of size 224×224×3, while almost none of the CBIS-DDSM images have this size (see Fig. 11), resizing is inevitable. However, the shape and orientation of the mass lesion in the image contains a lot of information about whether it is malignant or benign. Especially for images that were originally rectangular, resizing to square shaped images can come at a cost. Before delving into advanced resizing methods, one could consider how much effect the resizing has on the performance of the models by comparing the performance of originally square shaped images against originally rectangular images. If the performance is significantly worse on the originally rectangular shaped images, new considerations could be made for different resizing techniques. For instance, some techniques focus on resizing only the regions that are not of interest, therefore potentially preserving the original shape of the mass lesions. However, these techniques are often relatively hard to implement, reducing the simplicity of the method and increasing computational cost.

The last point of consideration for the data part is one not incorporated in this study but worth mentioning. In this study, the goal was to implement the final stage of the traditional CAD system. Stages such as detecting and segmenting the mass lesion in the full mammogram were beyond the scope and not necessary due to the convenience images. However, a portion of the literature on this subject does incorporate those preceding stages (see e.g. Al-antari Aisslab et al. (2018); Singh et al. (2019); Baccouche et al. (2022)). Even more so, they do not just create cropped versions of full mammograms, but use techniques to determine a contour around the mass lesion, putting every pixel outside that contour to zero, and only then crop the image. The resulting image is different from what was used in this study in the sense that the image displays more clearly the region of interest and the shape of the mass lesion. As we have seen in the incorrectly classified images, our model sometimes fails to identify the mass lesion in the image. Incorporating these detection and segmentation techniques might help resolve this issue and increase performance. In Appendix B, we show some examples of such images.

6.2 Lasso

Since we used one regressor for each pixel of the images, our logistic regression contained 50176 coefficients which had to be estimated using a much smaller total of 5424 images ($p \gg N$). For such cases, the MLE estimates are non-unique and the objective function has to be regularized. In this study, we used lasso (or l_1) regularization, since it conducts automatic model selection by shrinking coefficients to zero. If the data contains groups of strongly correlated regressors, lasso tends to select only a few out of the group by simply shrinking the coefficients of the others to zero. Additionally, if, for instance X_1 and X_2 are strongly correlated and have

a joint effect of 0.2, it does not make a difference whether lasso estimates β_1 and β_2 as 0.05 and 0.15. This selection process or shrinking process is arbitrary (Hastie et al. 2015). In our case, it is fair to assume that, for instance, pixels in regions with high intensity are highly correlated since they are linear combinations of each other. Therefore, to overcome the arbitrariness of the lasso, one could consider using additional regularization. One technique known to overcome the arbitrariness is ridge (or l_2) regularization. A combination of the two (often called elastic net), can be used to overcome both the $p \gg N$ and correlation issues.

6.3 Transfer Learning and Fine-Tuning

Incorporating transfer learning and fine-tuning is a very delicate process. First, one has to decide upon which layers to unfreeze for the network to sufficiently change its pre-trained parameters for the classification of the dataset of interest. The best amount of layers to unfreeze can dependent on the dataset and model used per task, but since there is no rule of thumb for the best procedure, it is often a case of iteratively evaluating what works best. In this study, we conducted such an iterative approach, but it was small due to the computational cost it comes with. Training neural networks is already computationally heavy, especially when using cross-validation as in this study. Also, incorporating an extensive unfreezing procedure would have become too computationally heavy. Therefore, we chose to focus more on an extensive cross-validation and only conducted a small iterative approach for deciding upon which layers to unfreeze. However, when computational cost is of no issue, one could consider incorporating both an extensive unfreezing search and cross-validation.

Secondly, apart from deciding which layers to unfreeze, a method for unfreezing has to be chosen. In this study, we chose to unfreeze the layers of residual block 5 all at once, since that is computationally the most inexpensive and easy to implement. However, methods such as gradual unfreezing, where a layer is unfrozen and trained, then a preceding layer is unfrozen and both are trained, etc., can be used as well. Alternatively, differential learning rate unfreezing can be used. In this method, earlier layer have different learning rates than deeper layers such that the earlier layers only slightly adjust their classification-general parameters, and deeper layers still more rigorously adjust their parameters. More methods can be chosen, but it might be interesting to consider them for future research.

6.4 Kernels

To create visualizations to what kernels in the network are most responsive to, gradient descent to optimize some loss function has to be performed. In other words, the visualizations are an estimation of what the kernels are most responsive to. In this study, we used a random initialized image, and adjusted this image with a step size of 0.1 over 200 iterations to get the kernel's responsiveness. Whether

this responsiveness is the desired maximum responsiveness is somewhat unclear. We tested for responsiveness by using a small range for both the step size and the amount of iterations, and visually evaluated whether the resulting visualization had clearer responsiveness. However, to quantitatively evaluate whether the visualization created the most responsive representation, one could look at the value of the loss function. In our case, we used the mean value of the activation map produced by the kernel as loss function. Higher mean value would indicate higher responsiveness.

6.5 Simulations

Since the amount of images contained in the CBIS-DDSM dataset is relatively limited for the optimization of deep CNN architectures such as ResNet50, both data augmentation and transfer learning were used. However, another consideration that can be made is the use of simulations. Techniques such as Monte Carlo simulations or synthetic data generation can be used to both address the data scarcity problem, but also to test the model's robustness by creating more diversity in the dataset. Since breast mass lesions have a wide variety of characteristics, diversity in the dataset is very important for the robustness of the model.

6.6 General lessons

To conclude this discussion, we end with some lessons learned from our methodology that we deem important for future work.

First, the quality of the dataset used to train deep CNN models is crucial for the model to learn relevant features of the images rather than memorizing noise. Therefore, techniques such as image enhancement, image augmentation, and normalization are important for the performance and robustness of the model.

Additionally, choosing the model's complexity for the task at hand is essential. Too complex models can overfit the data, especially when data scarcity is present, and can result in the model to learn noise instead of relevant features. ResNet50 potentially overcomes this problem by introducing residual blocks with skip connections, but regularization techniques such as dropout or l_2-regularization are still recommended to improve the model's generalizability.

Furthermore, for datasets with limiting samples such as the CBIS-DDSM dataset, transfer learning and fine-tuning can be a very powerful technique. Patterns learned in the earlier layers of pre-trained CNNs are still of great use when tested and evaluated on the smaller dataset of interest.

Also, in fields like medical diagnosis, the model's accuracy is not enough. Interpretability of the model's decision-making process by showing intermediate kernel representations or CAMs are very important to build trust with its users and to facilitate better understanding of the model.

Finally, for medical imaging classification, a variety of evaluation metrics is important. Not only insights in pure accuracy, but more informative insights like

sensitivity and specificity are important, as the costs of false negatives and false positive are very high.

7 Conclusion

Recently, deep learning models have demonstrated significant achievements in classifying and diagnosing breast masses within various CAD systems. The CNN architecture has been extensively adjusted and improved in numerous studies, often through integration with other methods like transfer learning and fine-tuning, to improve classification accuracy.

In this study, we have implemented the ResNet50 model for the classification of benign or malignant breast masses of cropped convenience images of the CBIS-DDSM dataset. The model was pre-trained using the large ImageNet dataset, and further trained on the CBIS-DDSM dataset using transfer learning and fine-tuning. Additionally, a lasso regularized logistic regression was trained and used as a baseline to compare with the proposed ResNet50 model. For both models, we used cross-validation to compute the optimal hyperparameters. Before training, the data was preprocessed using CLAHE image enhancement, image augmentation, and image resizing. For both models, we implemented a classification performance and visualization analysis.

The quantitative classification results showed a performance increase of around 20% for each of the evaluation metrics for the ResNet50 model, compared to the logistic regression. Specifically, the proposed ResNet50 model achieved an accuracy, sensitivity, and specificity of 79%, 73%, and 84% respectively, compared to 61%, 54%, and 65% respectively for the logistic regression. Both models achieved their highest performance for specificity and lowest performance for sensitivity, indicating both performed relatively better in identifying actual benign cases and relatively worse in identifying actual malignant cases. Additionally, for the logistic regression model, a pseudo-R^2 of 0.87 was found, indication the model showed an improved fit to the data compared to the null model without any regressors. Moreover, the lasso regularization resulted in a sparsity of 25.55%, indicating the regularization shrunk 25.55% of the coefficients to zero.

The heatmap of the coefficients of the logistic regression model showed no visual interpretable patterns. Specifically, the heatmap showed noise-like characteristics with alternating areas of negative and positive coefficients. Therefore, decisions made by the model could not be interpreted visually. For the ResNet50 model, the activations maps of each first convolutional layer of the residual blocks showed a universal CNN characteristic. The earlier layers showed indications of simple edge detectors in varying directions, with each layer becoming increasingly abstract until block 4, where the abstractness had increased to a point where visual interpretations were impossible. Then, for the kernel representations, similar result were found. In the earlier layers, the kernels are most responsive to color or simple directional edges.

The deeper into the network, the kernels showed responsiveness to an increasing amount of variation of patterns. Finally, from the class activation maps of both correctly and incorrectly classified images, it was shown that the ResNet50 model assigned the most intense regions of the images as most important when it correctly classified the images, while it did not assign those regions as important when it incorrectly classified the images.

Comparing the results of the preliminary methodology (see Section 3.12) and the main methodology, we observe that both the logistic regression and CNN model saw a decrease in classification performance when implemented on the CBIS-DDSM dataset. However, the decrease seen for the logistic regression was substantially bigger for the logistic regression than for the ResNet50 model. Additionally, while it was still possible to visually interpret the coefficients of the logistic regression model in the preliminary methodology, visual interpretation of the logistic model in the main methodology was not possible. This is contrary to what was seen for the visualization of the ResNet50 model. Apart from uninterpretable activation maps in deeper layers of the model, earlier activation maps, all kernel representations, and the class activation maps were still interpretable, making it easier to understand the model. Hence for easier classification tasks, where the images are easily separable by having very distinct shapes per class and are displayed at very similar regions of the image, both logistic regression and CNN architectures are adequate in terms of performance and interpretation. However, when the data becomes more complex with higher variety per class, the logistic regression becomes inadequate for both tasks, while the CNN architecture can be tuned to still be adequate for both.

Data & Code Availability

The MNIST dataset used in this study is a standard dataset contained in Google's Tensorflow 2.0 API. It can be easily extracted using simple Python. The CBIS-DDSM dataset used in this study is available in the Cancer Imaging Archive, https://wiki.cancerimagingarchive.net/pages/viewpage.action?pageId=22516629 or at https://www.kaggle.com/datasets/awsaf49/cbis-ddsm-breast-cancer-image-dataset. In this study the latter was used because the Cancer Imaging Archive has a 163GB DICOM version while the Kaggle website has a 6GB JPEG version.

The code written to conduct this study is publicly available for replication as a git repository on GitHub at https://github.com/siemfenne/master_thesis. Recommended is to first thoroughly read the readme file to avoid possible issues.

References

Abdelhafiz, D., Bi, J., Ammar, R., Yang, C., and Nabavi, S. (2020). Convolutional neural network for automated mass segmentation in mammography. *BMC Bioinformatics*, 21.

Al-antari Aisslab, M. A., Al-masni, M., Choi, M.-T., Han, S.-M., and Kim, T.-S. (2018). A fully integrated computer-aided diagnosis system for digital x-ray mammograms via deep learning detection, segmentation, and classification. *International Journal of Medical Informatics*, 117.

Al-masni, M., Al-antari Aisslab, M. A., Park, J.-m., Gi, G., Kim, T.-Y., Rivera, P., Valarezo Añazco, E., Choi, M.-T., Han, S.-M., and Kim, T.-S. (2018). Simultaneous detection and classification of breast masses in digital mammograms via a deep learning yolo-based cad system. *Computer Methods and Programs in Biomedicine*, 157.

Alkhaleefah, M., Ma, S.-C., Chang, Y.-L., Huang, B., Kumar, P., and Achhannagari, V. (2020). Double-shot transfer learning for breast cancer classification from x-ray images. *Applied Sciences*, 10:3999.

Baccouche, A., Zapirain, B., and Elmaghraby, A. (2022). An integrated framework for breast mass classification and diagnosis using stacked ensemble of residual neural networks. *Scientific Reports*, 12:12259.

Bengio, Y., Lamblin, P., Popovici, D., Larochelle, H., and Montreal, U. (2007). Greedy layer-wise training of deep networks. volume 19.

Bi, W. L., Hosny, A., Schabath, M., Giger, M., Birkbak, N., Mehrtash, A., Allison, T., Arnaout, O., Abbosh, C., Dunn, I., Mak, R., Tamimi, R., Tempany, C., Swanton, C., Hoffmann, U., Schwartz, L., Gillies, R., Huang, R., and Aerts, H. (2019). Artificial intelligence in cancer imaging: Clinical challenges and applications. *CA: A Cancer Journal for Clinicians*, 69.

Chakravarthy, S. and Rajaguru, H. (2021). Automatic detection and classification of mammograms using improved extreme learning machine with deep learning. *IRBM*, 43.

Chollet, F. (2017). *Deep Learning with Python*. Manning.

Cybenko, G. (1989). Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals and Systems*, 2:303–314.

Defazio, A., Bach, F., and Lacoste-Julien, S. (2014). Saga: A fast incremental gradient method with support for non-strongly convex composite objectives. In Ghahramani, Z., Welling, M., Cortes, C., Lawrence, N., and Weinberger, K.,

editors, *Advances in Neural Information Processing Systems*, volume 27. Curran Associates, Inc.

Dhahri, H., Rahmany, I., Mahmood, A., Al Maghayreh, E., and Elkilani, W. (2020). Tabu search and machine-learning classification of benign and malignant proliferative breast lesions. *BioMed Research International*, 2020:1–10.

Dixon, M., Klabjan, D., and Bang, J. H. (2016). Classiffication-based financial markets prediction using deep neural networks. *CoRR*, abs/1603.08604.

Efron, B. and Hastie, T. (2016). *Computer Age Statistical Inference: Algorithms, Evidence, and Data Science*. Cambridge University Press, USA, 1st edition.

Elmore, J., Jackson, S., Abraham, L., Miglioretti, D., Carney, P., Geller, B., Yankaskas, B., Kerlikowske, K., Onega, T., Rosenberg, R., Sickles, E., and Buist, D. (2009). Variability in interpretive performance at screening mammography and radiologists' characteristics associated with accuracy 1. *Radiology*, 253:641–51.

Eltrass, A. and Salama, M. (2020). A new fully automated scheme for computer-aided detection and breast cancer diagnosis using digitized mammograms. *IET Image Processing*, 14.

Ferlay, J., Colombet, M., Soerjomataram, I., Parkin, D., Piñeros, M., Znaor, A., and Bray, F. (2021). Cancer statistics for the year 2020: An overview. *International Journal of Cancer*, 149.

Fuleky, P. (2019). *Macroeconomic Forecasting in the Era of Big Data: Theory and Practice*. Advanced Studies in Theoretical and Applied Econometrics. Springer International Publishing.

Garcia-Gasulla, D., Parés, F., Vilalta, A., Moreno, J., Ayguadé, E., Labarta, J., Cortés, U., and Suzumura, T. (2017). On the behavior of convolutional nets for feature extraction. *CoRR*, abs/1703.01127.

Glorot, X. and Bengio, Y. (2010). Understanding the difficulty of training deep feedforward neural networks. *Journal of Machine Learning Research - Proceedings Track*, 9:249–256.

Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep Learning*. MIT Press. http://www.deeplearningbook.org.

Halevy, A., Norvig, P., and Fernando, N. (2009). The unreasonable effectiveness of data. *Intelligent Systems, IEEE*, 24:8 – 12.

Hamed, G. (2020). Yolo based breast masses detection and classification in full-field digital mammograms. *Computer methods and programs in biomedicine*, 200.

Hastie, T., Tibshirani, R., and Wainwright, M. (2015). *Statistical Learning with Sparsity: The Lasso and Generalizations*. Chapman & Hall/CRC.

He, K., Zhang, X., Ren, S., and Sun, J. (2015). Deep residual learning for image recognition.

He, K., Zhang, X., Ren, S., and Sun, J. (2016). Identity mappings in deep residual networks. volume 9908, pages 630–645.

Hornik, K. (1991). Approximation capabilities of multilayer feedforward networks. *Neural Networks*, 4(2):251–257.

James, G., Witten, D., Hastie, T., and Tibshirani, R. (2013). *An Introduction to Statistical Learning: with Applications in R*. Springer.

Jiao, Z., Gao, X., Wang, Y., and L, J. (2017). A parasitic metric learning net for breast mass classification based on mammography. *Pattern Recognition*, 75.

Jobson, D. J., ur Rahman, Z., and Woodell, G. A. (1997). Properties and performance of a center/surround retinex. *IEEE transactions on image processing : a publication of the IEEE Signal Processing Society*, 6 3:451–62.

Kiefer, J. and Wolfowitz, J. (1952). Stochastic Estimation of the Maximum of a Regression Function. *The Annals of Mathematical Statistics*, 23(3):462 – 466.

Knight, K. and Fu, W. (2000). Asymptotics for lasso-type estimators. *Annals of Statistics*, 28.

Lee, R., Gimenez, F., Hoogi, A., Miyake, K., Gorovoy, M., and Rubin, D. (2017). A curated mammography data set for use in computer-aided detection and diagnosis research. *Scientific Data*, 4:170177.

Li, C., Guo, J., Porikli, F., and Pang, Y. (2018). Lightennet: A convolutional neural network for weakly illuminated image enhancement. *Pattern Recognition Letters*, 104.

Li, S., Song, W., Fang, L., Chen, Y., Ghamisi, P., and Benediktsson, J. A. (2019). Deep learning for hyperspectral image classification: An overview. *IEEE Transactions on Geoscience and Remote Sensing*, 57(9):6690–6709.

Liu, D. C. and Nocedal, J. (1989). On the limited memory bfgs method for large scale optimization. *Math. Program.*, 45(1–3):503–528.

Maas, A. L. (2013). Rectifier nonlinearities improve neural network acoustic models.

Muramatsu, C., Nishio, M., Goto, T., Oiwa, M., Morita, T., Yakami, M., Kubo, T., Togashi, K., and Fujita, H. (2020). Improving breast mass classification by shared data with domain transformation using a generative adversarial network. *Computers in Biology and Medicine*, 119:103698.

Nair, V. and Hinton, G. E. (2010). Rectified linear units improve restricted boltzmann machines. In *ICML 2010*, pages 807–814.

Nasir Khan, H., Shahid, A., Raza, B., Dar, A., and Alquhayz, H. (2019). Multi-view feature fusion based four views model for mammogram classification using convolutional neural network. *IEEE Access*, PP:1–1.

National Cancer Institute (2023). Cancer stat facts: Female breast cancer. https://seer.cancer.gov/statfacts/html/breast.html.

Olgac, A. and Karlik, B. (2011). Performance analysis of various activation functions in generalized mlp architectures of neural networks. *International Journal of Artificial Intelligence And Expert Systems*, 1:111–122.

Oza, P., Sharma, P., Patel, S., Adedoyin, F., and Bruno, A. (2022). Image augmentation techniques for mammogram analysis. *Journal of Imaging*, 8:141.

Perez, L. and Wang, J. (2017). The effectiveness of data augmentation in image classification using deep learning. *CoRR*, abs/1712.04621.

Pisano, E., Zong, S., Hemminger, B., DeLuca, M., Johnston, R., Muller, K., Braeuning, M., and Pizer, S. (1998). Contrast limited adaptive histogram equalization image processing to improve the detection of simulated spiculations in dense mammograms. *Journal of digital imaging*, 11(4):193—200.

Pizer, S. M., Amburn, E. P., Austin, J. D., Cromartie, R., Geselowitz, A., Greer, T., ter Haar Romeny, B., Zimmerman, J. B., and Zuiderveld, K. (1987). Adaptive histogram equalization and its variations. *Computer Vision, Graphics, and Image Processing*, 39(3):355–368.

Polyak, B. (1964). Some methods of speeding up the convergence of iteration methods. *Ussr Computational Mathematics and Mathematical Physics*, 4:1–17.

Qi, Y., Yang, Z., Sun, W., Lou, M., Lian, J., Zhao, W., Deng, X., and Ma, Y. (2021). A comprehensive overview of image enhancement techniques. *Archives of Computational Methods in Engineering*, 29:583–607.

Ragab, D. A., Sharkas, M. A., Marshall, S., and Ren, J. (2019). Breast cancer detection using deep convolutional neural networks and support vector machines. *PeerJ*, 7.

Ragab, M., Albukhari, A., Alyami, J., and Mansour, R. (2022). Ensemble deep-learning-enabled clinical decision support system for breast cancer diagnosis and classification on ultrasound images. *Biology*, 11:439.

Sahakyan, A. and Sarukhanyan, H. (2012). Segmentation of the breast region in digital mammograms and detection of masses. *International Journal of Advanced Computer Science and Applications - IJACSA*, 3.

Santos, I., Castro, L., Rodriguez-Fernandez, N., Torrente-Patino, A., and Carballal, A. (2021). Artificial neural networks and deep learning in the visual arts: A review. *Neural Computing and Applications*, 33:121–157.

Selvaraju, R. R., Das, A., Vedantam, R., Cogswell, M., Parikh, D., and Batra, D. (2016). Grad-cam: Why did you say that? visual explanations from deep networks via gradient-based localization. *CoRR*, abs/1610.02391.

Shams, S., Platania, R., Zhang, J., Kim, J., and Park, S.-j. (2018). *Deep Generative Breast Cancer Screening and Diagnosis*, pages 859–867.

Shen, L., Margolies, L., Rothstein, J., Fluder, E., McBride, R., and Sieh, W. (2019). Deep learning to improve breast cancer detection on screening mammography. *Scientific Reports*, 9:1–12.

Shewchuk, J. R. (1994). An introduction to the conjugate gradient method without the agonizing pain. Technical report, USA.

Shijie, J., Ping, W., Peiyi, J., and Siping, H. (2017). Research on data augmentation for image classification based on convolution neural networks. pages 4165–4170.

Shorten, C. and Khoshgoftaar, T. M. (2019). A survey on image data augmentation for deep learning. *J. Big Data*, 6:60.

Siegel, R. L., Miller, K. D., Fuchs, H. E., and Jemal, A. (2022). Cancer statistics, 2022. *CA: A Cancer Journal for Clinicians*, 72(1):7–33.

Singh, V. K., Rashwan, H., Romaní, S., Akram, F., Pandey, N., Sarker, M. M. K., Saleh, A., Arenas, M., Arquez, M., Puig, D., and Torrents-Barrena, J. (2019). Breast tumor segmentation and shape classification in mammograms using generative adversarial and convolutional neural network. *Expert Systems with Applications*, 139:112855.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2014). Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15:1929–1958.

Stock, J. and Watson, M. (2019). *Introduction to Econometrics*. Pearson series in economics. Pearson.

Sun, C., Shrivastava, A., Singh, S., and Gupta, A. (2017). Revisiting unreasonable effectiveness of data in deep learning era.

Szegedy, C., Ioffe, S., Vanhoucke, V., and Alemi, A. (2016). Inception-v4, inception-resnet and the impact of residual connections on learning. *AAAI Conference on Artificial Intelligence*, 31.

Tsochatzidis, L., Koutla, P., Costaridou, L., and Pratikakis, I. (2021). Integrating segmentation information into cnn for breast cancer diagnosis of mammographic masses. *Computer Methods and Programs in Biomedicine*, 200:105913.

van de Geer, S., Bühlmann, P., Ritov, Y., and Dezeure, R. (2013). On asymptotically optimal confidence regions and tests for high-dimensional models. *The Annals of Statistics*, 42.

Wang, Q. and Yuan, Y. (2014). High quality image resizing. *Neurocomputing*, 131:348–356.

Zhang, C.-H. and Zhang, S. (2011). Confidence intervals for low-dimensional parameters in high-dimensional linear models. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 76.

Zhang, Q., Li, Y., Zhao, G., Man, P., Lin, Y., and Wang, M. (2020). A novel algorithm for breast mass classification in digital mammography based on feature fusion. *Journal of Healthcare Engineering*, 2020:1–11.

Zhou, S., Gan, J., Xu, L.-D., and John, R. (2007). Interactive image enhancement by fuzzy relaxation. *International Journal of Automation and Computing*, 4:229–235.

Zuiderveld, K. J. (1994). Contrast limited adaptive histogram equalization. In *Graphics gems*.

A Monte Carlo Simulation

To investigate the performance and consistency of the logistic regression coefficients under controlled conditions, we conducted a Monte Carlo simulation. The process for one iteration of the simulation was as follows. First, we chose a fixed coefficient vector $\beta = (\beta_1, \ldots, \beta_d)$ (the same vector was used for all iterations). Then, we randomly initialized 14,000 28×28 images, with each pixel in the randomized image being drawn randomly from a uniform distribution on the interval [0, 255]. This was followed by creating 14,000 binary outcomes using the fixed coefficient vector and the randomized images in the form of the logistic regression model in Eq. (3). Finally, the artificially generated binary outcomes and the randomized images were used to estimate the coefficients back, here denoted as $\hat{\beta}$. After estimation, we computed the mean absolute difference (MAD) between the coefficients of the fixed coefficient vector and the coefficients of the estimated coefficient vector. In other words, we computed:

$$\text{MAD} = \frac{1}{d} \sum_{j=1}^{d} |\beta_j - \hat{\beta}_j|. \tag{35}$$

We conducted the simulation 100 times and computed the mean of the mean absolute differences (MMAD) and the standard deviation of the mean absolute differences (SMAD). We observed values of 0.036 and 0.005 for the MMAD and SMAD respectively against an average magnitude of 0.048 for our fixed coefficient vector. With the MMAD being less than the average magnitude of the fixed coefficient vector, and the SMAD being very small, the result suggests the estimated coefficients are not only consistent but also relatively accurate in magnitude compared to the original coefficients.

To further study the logistic regression coefficients under controlled conditions, we also plot the distribution of the MAD values of each iteration, see Fig. 21. From the distribution, we again observe that the estimated coefficients are consistently estimated with similar magnitude throughout the iterations, showing no big outliers and most iterations centered around the MMAD.

Since in our analysis used in the logistic regression MNIST classification section of our study, we used visual representations of the coefficients, we also conducted a visual analysis in our Monte Carlo simulation. To do so, we conveniently chose our fixed 784-dimension coefficient vector to be a recognizable pattern when transformed back to a 28×28 image. The idea was then to keep score of each of the 100 estimated coefficient vectors, and in the end take the average estimated coefficient vector. If the logistic regression is able to consistency estimate the fixed coefficient vector back, we should be able to see a similar pattern as the fixed coefficient vector with similar magnitudes. The results are shown in Fig. 22. The left panel shows the fixed coefficient vector as a 28×28 image. For convenience, we used a zero digit as our recognizable pattern. The right panels show what the logistic regression on average estimated back. We observe that the two images are visually very hard to distinguish from each other, other than a very small difference magnitude. The color bar of the right panel has a slightly bigger range.

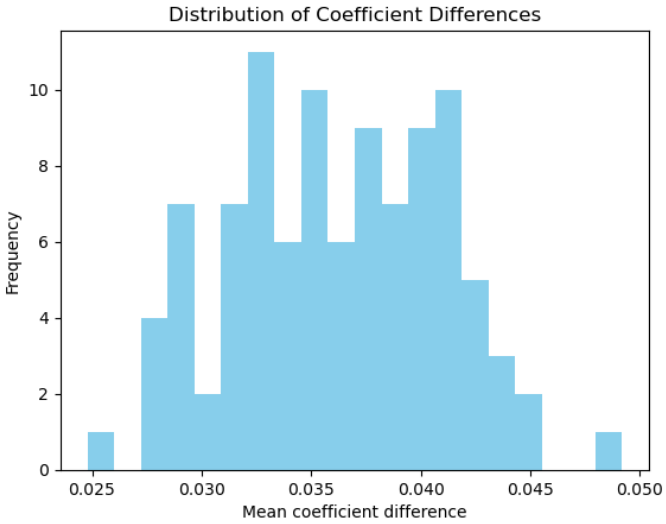


Figure 21: Distribution of mean absolute differences (MAD) between the fixed coefficient vector and estimated coefficient vector for 100 Monte Carlo iterations. No big outliers are seen, and most of the MAD are centered around the mean, indicating the logistic regression is able to consistently estimate the fixed coefficient vector back.

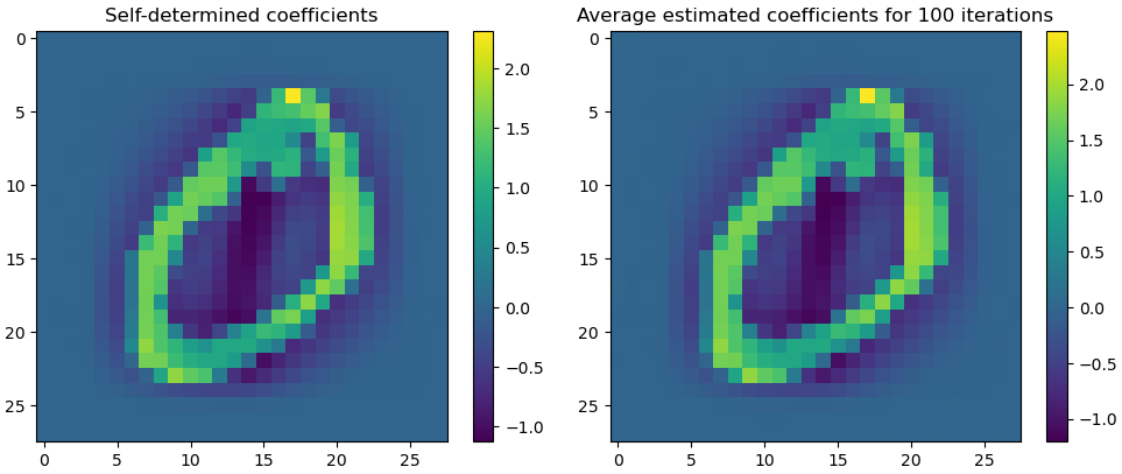


Figure 22: Left panel: the conveniently chosen fixed coefficient vector, shown as a 28×28 image. Right panel: the average estimated coefficient vector, shown as a 28×28 image. We observe that the two panels are visually very hard to distinguish, only having a slight difference in magnitude. Again, this is evidence that the logistic regression is able to consistently estimate the fixed coefficient vector back.

B Detected and Segmented ROI masses

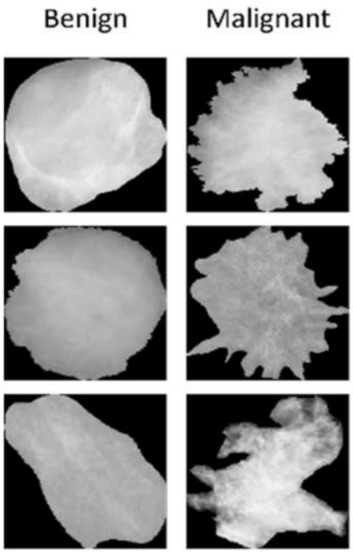


Figure 23: Samples of detected and segmented ROI masses of the CBIS-DDSM dataset. A portion of the literature on CAD systems incorporates detecting and segmentation stages, resulting in cropped images with clearer ROIs and shapes than the images used in this study. These images could help the model identify the mass lesion easier and increase classification performance. This figure was taken from Baccouche et al. (2022).