Deep Learning for Early Detection of Breast Cancer: Exploring Transfer Learning and CNNs

Alexandra Benova, Christina Gollub and Lisa Mesch

The final project: Breast Cancer Detection with a Basic CNN Model and Transfer Learning Models

Implementing Artificial Neural Networks with Tensorflow
University of Osnabrück
WINTERSEMESTER 2022/23

Breast Cancer Detection

Alexandra Benova	995241
Christina Gollub	996953
Lisa Mesch	996140

Abstract

Breast cancer is a prevalent and significant health concern worldwide, with mammography testing being the most effective tool for early detection. However, even with highly skilled radiologists examining mammograms, the rate of missed breast cancers remains high due to human error, distraction, and fatigue. As a team of three women, we have chosen to focus on breast cancer detection because we believe it is a crucial and compelling topic. Our motivation behind this project is to explore techniques for analyzing mammograms that can accurately detect and prevent late or unnecessary treatments. By expanding on previous research using deep learning models to detect and classify breast cancer, we hope to contribute to the development of accurate and efficient deep learning models for breast cancer detection. Our ultimate goal is to improve the chances of successful treatment and save lives. We will do this by exploring different architectures, like the ResNet50, VGG16 and InceptionV3, used with a transfer learning method. These models are compared with each other and a basic convolutional neural network.

Keywords: Breast Cancer Detection, CNN, Transfer Learning, ResNet50, VGG16, InceptionV3

Contents

Chapter 1

Motivation

1.1 Motivation

Breast cancer is one of the most common oncological diseases among women worldwide. With it being the 3rd most common cancer in Bulgaria [18], accounting approximately 30 percent of all cancer cases in Germany [10] and ranking number one in the UK with approximately 57000 new cases each year [20], breast cancer remains a significant health concern.

Detecting breast cancer early through screening tests is crucial to improve the chances of successful treatment. Mammography is the most effective tool for early detection. However, even with highly skilled radiologists examining mammograms, the rate of missed breast cancers are high. Outside factors, such as distraction, fatigue and human error must be minimised to ensure the accuracy. Therefore, researchers have been focusing on developing Artificial Intelligence Agents to minimise wrong interpretations. The probability of a false-positive mammography screening in Europe can range from 8 to 21 percent.[24]

As a team of three women, we have chosen to focus on breast cancer detection, as we believe it is a highly significant and compelling topic. The motivation behind this project is to explore techniques for analysing mammograms that can accurately detect and prevent late or unnecessary treatments. In the following project, we aim to compare the performance of a basic convolutional neural network architecture and three different models, ResNet50, VGG16 and InceptionV3, used for our Transfer learning method, with further implementations. All of the models are trained on "Breast Histopathology Images" from Kaggle [14]. The images depict whole mount slides images of the most common form of breast cancer, invasive ductal carcinoma. Our project is based on previous research using deep learning models to detect and classify breast cancer. The studies "Transfer Learning and Fine Tuning in Breast Mammogram Abnormalities Classification on CBIS-DDSM Database" Falconi[9] at al. by and "Breast Cancer Detection in Mammograms using Deep Learning Techniques" by Adam Jaamour[2], amongst others, show promising results in improving breast cancer detection. Our motivation behind this project is that we hope to contribute to the development of accurate and efficient deep learning models for breast cancer detection by expanding on these studies and investigating the use of transfer learning.

Chapter 2

Introduction

In recent years, deep learning models have demonstrated exceptional achievements in a variety of medical imaging tasks, including breast cancer detection and classification. [4] As a result, we chose this project to investigate the application of transfer learning in the development of deep learning models for breast cancer detection. Our goal is to construct a convolutional neural network architecture and transfer learning models and then evaluate their performance. Additionally, we aim to compare the accuracy of these models to other approaches to obtain a comprehensive evaluation.

2.1 Our approach

2.1.1 Basic CNN

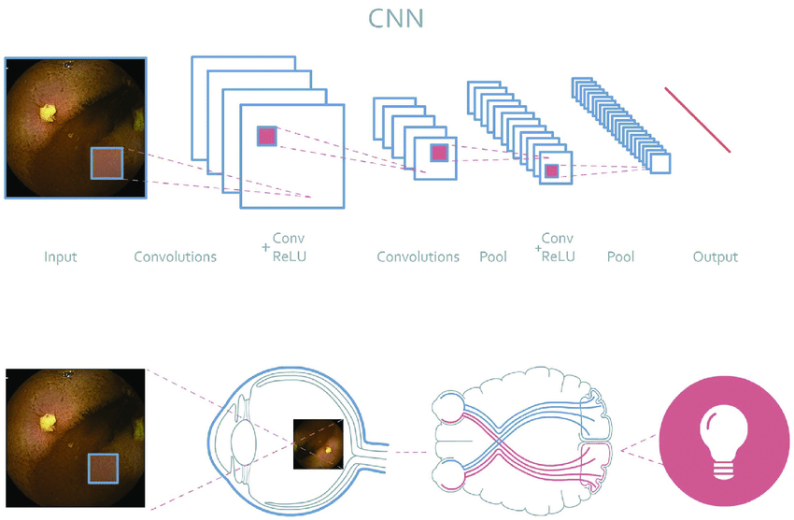

Figure 2.1: Schematic representation of the analogy between a CNN and a human visual cortex

Convolutional Neural Networks (CNNs) are a type of Neural Network which have a unique architecture inspired by the human visual cortex. By having local receptive fields that only respond to specific regions of visual stimuli, CNNs are highly efficient at performing complex visual tasks. They are primarily used for image processing tasks, such as image classification or object detection. In contrast

to traditional neural networks, CNNs are only partially connected, which allows them to work quickly with large images. To simplify the input images, CNNs stack convolutional and pooling layers together before processing the information with a shallow artificial neural network for classification. [25] The core component of a CNN is the convolutional layer. It applies a set of filters to the image to extract features and each filter generates a feature map, which represents a particular attribute in the image. Each layer has multiple filters, allowing different features to be detected simultaneously. Pooling layers are implemented to reduce the spatial dimensionality of the image while preserving its essential features. This reduces the load on the GPU. These layers help to reduce the number of parameters and prevent overfitting. At the end of the convolutional and pooling layer stack, a fully connected MLP is placed. Our basic CNN is built up of four convolutional layers with increasing filter size, followed by two fully connected layers and an output layer. CNNs have achieved remarkable success in various domains, including medicine. They can process large images much faster than traditional machine learning methods.

2.1.2 Transfer Learning

Transfer learning is a widely used technique in deep learning that involves the transfer of learned features, such as weights and biases, from a pre-trained model to a new network model for testing. Pre-trained models have already been trained on similar domains, making them an effective starting point for new models. With a wide range of pre-trained architectures available, utilising these models can save significant time and computational power that would otherwise be required to train large models on extensive datasets. Additionally, training a network from scratch can be a time-consuming process, taking weeks or even months to complete. However, by leveraging pre-trained weights and biases, the learning process can be significantly sped up, allowing for quicker and more efficient training of new networks. Therefore, the use of pre-trained models has become an essential technique in deep learning, enabling faster and more accurate development of new models. [8] We will use three different kind of pre-trained models to explore the transfer learning technique: VGG16, ResNet50 and InceptionV3.

VGG 16

VGG16 is a powerful convolutional neural network that has gained widespread use in computer vision tasks, including object detection and image classification. This model has gained popularity as one of the best computer vision models available. The creators of VGG16 improved on prior-art configurations by evaluating and increasing the depth of the network using small (3×3) convolution filters. This led to a significant improvement in performance, with 16-19 weight layers and approximately 138 trainable parameters.[23]

VGG16 has 16 weight layers and 21 layers in total. It uses input tensors of size 224 or 244 with 3 RGB channels. VGG16 stands out for using 3x3 filters with stride 1 and consistent padding, along with 2x2 max pool layers. It has a consistent arrangement of convolution and max pool layers throughout the architecture. The number of filters increases from 64 to 512 as you move through the layers. VGG16 also has three fully-connected layers, with the last layer being a softmax layer for classification.

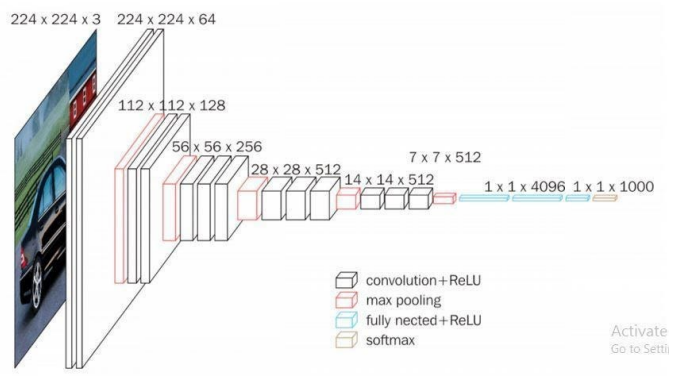

Figure 2.2: : Schematic representation of a VGG16

VGG16 is widely used for object detection and classification, with the ability to classify 1000 images of 1000 different categories with an accuracy of 92.7% .[27] It is particularly well-suited for image classification and is often used in transfer learning due to its ease of use. In the context of breast cancer detection, VGG16's high accuracy and ability to recognize features in images make it an ideal option for analyzing mammograms and identifying potential cancerous cells.

Residual Neural Networks

By increasing the depth of a network, so called deep CNNs are able to learn more features. However, this can cause vanishing gradients and degradation. Therefore, the residual network was created by a Microsoft team.[26] Residual Networks or ResNet for short, is a deep learning model that is designed to address the problems occurring in a deep neural network. The network was inspired by VGG neural networks, but with fewer filters and lower complexity.[5] The "Residual Block" with the "skip connection" from the ResNet adds the output from the previous layer to the layer ahead.[26] The Microsoft team were able to create ResNet18, -34, -50, 101 and 152. Through this technique, a neural network with 152 layers was able to be trained while still having a lower complexity than VGGs.[6] ResNet has been shown to be highly effective in a range of image classification tasks, including medical image analysis. Its ability to learn features at multiple levels of abstraction led to its success, especially where the features that indicate disease may be subtle. In the context of the breast cancer image detection, ResNet allows for detecting complex features in images which could be critical for identifying subtle patterns of cancer. Therefore, ResNet can improve the accuracy and effectiveness of detecting breast cancer. This could potentially lead to better outcome and earlier detection.[3] ResNet50 is a more complex and deeper residual neural network. As the name indicates, it is a ResNet architecture build up of 50 layers. The main difference to the previous ResNet architecture, the ResNet-34, is the bottleneck architecture. Each of the 2-layer blocks in ResNet34 were replaced by a 3-layer bottleneck block, creating a higher accuracy and achieving a better performance rate.[5]

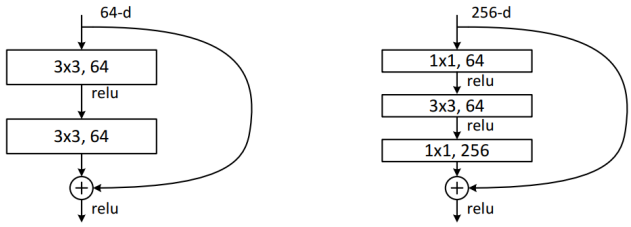

Figure 2.3: : Left: a building block for ResNet34. Right: a "bottleneck" building block for ResNet-50

Inception V3

The Inception V3 model is another deep convolutional neural network (CNN) architecture used in computer vision tasks, particularly in image recognition and classification. Developed by Google researchers in 2015 [15], Inception V3 is an extension of the earlier Inception model, designed to address some of its limitations, such as high computational requirements and overfitting. Inception V3 uses a combination of convolutional and pooling layers with various filter sizes and depths to extract features from images, followed by global average pooling and a softmax output layer for classification. With its improved accuracy and efficiency, Inception V3 has become a popular choice for image-related applications, including object detection, segmentation, and transfer learning.[15] The diagram of the Inception V3 model

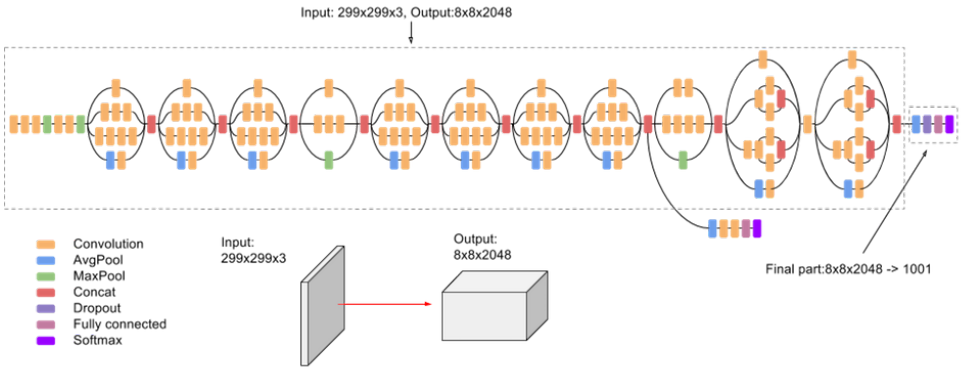

Figure 2.4: A high-level diagram of the InceptionV3 model

shows the different parts of the model. It starts with the input image, which goes through a series of convolutional layers, including factorized convolutions. The intermediate layers also include auxiliary classifiers to improve the training process. Finally, the model ends with fully connected layers and a softmax activation function for image classification. The diagram provides a visual representation of the complex architecture of the Inception V3 model and how the different layers work together to process and classify images.

2.1.3 Dataset

The selection of an appropriate dataset is a crucial step in developing machine learning models that are both accurate and robust. This is a complex process

that involves balancing computational limitations with the need for high-quality and representative data. Identifying a dataset that is sufficiently large, diverse, and relevant to the research question at hand is often a challenging task. To aid in the development of machine learning algorithms for the automated detection of Invasive Ductal Carcinoma (IDC), a prevalent subtype of breast cancer, the breast histopathology images dataset from Paul Mooney on Kaggle [14] was chosen.

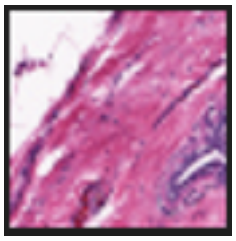

Figure 2.5: One of the patches labelled 10253_idx5_x1001_y1251_class0

These images are of the most prevalent subtype of breast cancer, Invasive Ductal Carcinoma (IDC), and consist of 277,524 patches that are 50 x 50 in size. These patches were extracted from a larger dataset of 162 whole mount slides, and were specifically chosen to focus on regions containing the IDC, which is of particular interest to pathologists.

Out of the total number of patches, 198,738 were negative for IDC, while 78,886 were positive. Although the dataset is unbalanced with more negative patches than positive patches for IDC, it can still be a valuable resource for developing deep learning models. The file names of each patch contain useful information about the patient ID, the specific x- and y-coordinates from which the patch was cropped, and whether the patch contains IDC or not. For example, a sample file name is "10253 idx5 x501 y351 class1.png".

While not the largest dataset available, the breast histopathology images dataset provides a good representation of the task at hand and includes a relatively balanced number of IDC-positive and IDC-negative patches. This balanced nature is essential for the development of accurate classification models. Despite these benefits, it is important to note that every dataset has its limitations. The representativeness of the data, inherent biases, and other confounding factors can all affect the reliability and generalizability of the results obtained from this dataset.

Therefore, it is essential for us as researchers to carefully consider the limitations of this dataset and continue to explore and incorporate new data sources to improve the accuracy and robustness of machine learning models for breast cancer diagnosis.

Chapter 3

Implementation

3.1 Data extraction/ preparation

This research project aims to improve the accuracy of a breast cancer diagnosis model through image data augmentation and preprocessing. The dataset used consists of breast cancer histopathology images, divided into two classes: IDC(+) and IDC(-), corresponding to the presence and absence of invasive ductal carcinoma, respectively.

The code initiates by importing essential libraries like NumPy, Matplotlib, and Keras. Image augmentation is carried out utilizing the ImageDataGenerator function from Keras. The function applies different image transformations like rotation, shifting, and flipping to magnify the size of the dataset.[22]

```
datagen = ImageDataGenerator(
    rotation_range=20,
    width_shift_range=0.1,
    height_shift_range=0.1,
    zoom_range=0.1,
    horizontal_flip=True,
    vertical_flip=True,
    fill_mode='nearest'
)
```

The dataset is then partitioned into positive and negative categories based on the image filenames, and the count of images in each class is displayed. In the beginning this posed a challenge, however with learning new techniques, we were able to overcome it.

```
neg_img = []
pos_img = []

for img in breast_img:
    if img[-5] == '0' :
        neg_img.append(img)

    elif img[-5] == '1' :
        pos_img.append(img)
```

```
neg_num = len(neg_img)
pos_num = len(pos_img)

total_img_num = neg_num + pos_num

print('Number of Images in IDC (-): {}' .format(neg_num))
print('Number of Images in IDC (+) : {}' .format(pos_num))
print('Total Number of Images : {}' .format(total_img_num))
```

In order to gain a deeper understanding of the dataset, we utilized Matplotlib to exhibit some sample images from both positive and negative classes. By visually inspecting the images, we were able to get a better sense of the features that distinguish between the two classes. This allowed us to develop a better intuition for the dataset, and to identify potential challenges that might arise during model development.

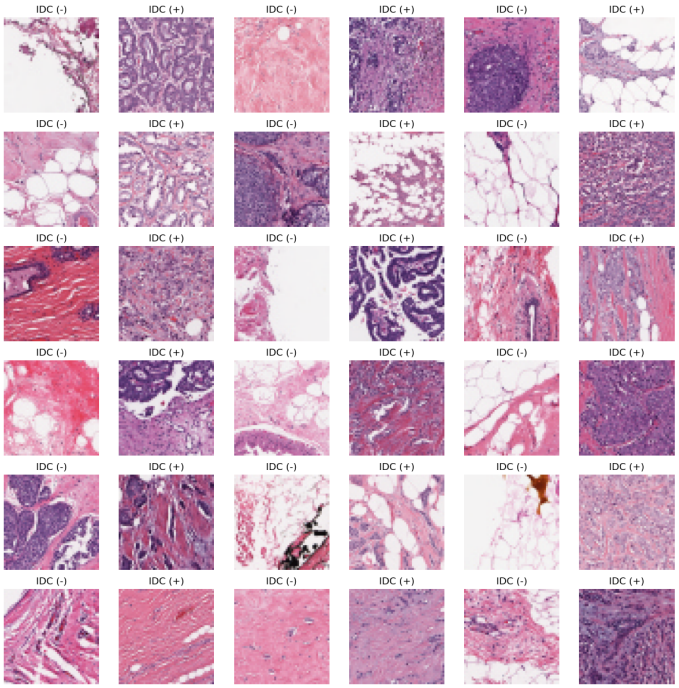

Figure 3.1: : Matplotlib output exhibiting sample images of IDC(+) and IDC(-)

Subsequently, OpenCV is employed to load and resize the images, and the features and labels are merged and randomly shuffled to prepare them for machine learning model training. Ultimately, the features are transformed into a NumPy array to facilitate further processing.

This image data augmentation and preprocessing code is a crucial step towards improving the accuracy of the breast cancer diagnosis model.

As we proceeded with implementing the different models, we became aware of the unbalanced nature of the dataset. Therefore, we believe it is crucial to acknowledge this limitation of the dataset and consider it during our analysis and interpretation of the results.

3.2 CNN Model

The Convolutional Neural Network (CNN) model is created in Keras and is defined as a sequential model. This means that the layers are added to the model in sequence. The layers are Conv2D, MaxPooling2D, Dropout and Dense. The architecture consists of several convolutional and pooling layers which are followed by a few dense layers.

```
model = Sequential()

model.add(Conv2D(32, (3, 3), padding = 'same', activation = 'relu', input_shape = (50, 50, 3)))
model.add(MaxPooling2D(2, 2))
model.add(Dropout(0.25))
```

The convolutional layers are added to the model by Conv2D layers with a 32 filter of size (3,3) and a padding of "same". This first layer applies 32 filters to the input image and generates 32 feature maps. That the padding parameter is set to same, means that the size of the output feature map is the same as the input size. The input images are of size 50x50 with three colour channels, which is set with the input_shape parameter. In this layer, the activation function is ReLU, which helps to introduce non-linearity into the model. After the "Conv2D" layer, a "MaxPooling2D" layer is implemented. This layer adds a pooling layer to the model with a pool size of (2,2), which helps to downsample the input feature maps. The input images are reduced in size by a factor of two. The purpose of this layer is to reduce the spatial size of the feature maps and to help the model generalise better. On top of this, the dropout layer is added to the model as a regularisation technique. "Dropout" randomly sets a fraction of the input units to zero during training, which helps to prevents overfitting. In this case the regularisation layer is added to the model with a rate of 0.25. This means that 25% of the input units are randomly dropped during training. To create deeper and more complex layers, the above operations are repeated with different size and numbers of filters. The following "Conv2D" layer has 64 filters with the same kernel size and padding. This layer generates 64 feature maps and applies the ReLU activation function as well. Another identical Maxpooling layer and Dropout layer are added as the previous layers.

```
model.add(Flatten())
model.add(Dense(128, activation = 'relu'))

model.add(Dropout(0.5))
model.add(Dense(2, activation = 'sigmoid'))
```

After the final convolutional layer, a "Flatten" layer is added to the model. The Flatten layer is used to convert the output of the convolutional layers from a 3D output into a 1D vector, which is then fed into a dense layer, which is fully connected. They have 128 neurons and a ReLU activation function. In addition, another "Dropout" layer is added to the model to prevent overfitting. Finally, a "Dense" layer with two units and a sigmoid activation function is added as the output layer of the model. This layer generates the output probabilities for each class. The sigmoid activation function ensures that the output values are between zero and one, which are the probabilities of each class. Overall, the CNN architecture is designed to learn and extract features from the input images using multiple convolutional, pooling, fully connected and output layers.

3.2.1 Optimisation and Hyperparameters

The use of the dropout and data augmentation techniques helps to prevent overfitting and improve the performance of the model. Overfitting is a common problem in deep neural networks. There are many techniques to conquer it, which will be highlighted throughout the paper. For our CNN, we implemented the dropout layer and data augmentation. Dropout randomly deactivates some of the neurons during training to force the model to learn about generalised features. The technique of applying a dropout with the rate of 0.25 after each pooling layer, has been shown to improve the performance of a deep learning model. Data Augmentation involves changing the orientation of images, selecting a part of an image randomly. In the case of our CNN, the data augmentation improved our results by achieving a higher validation accuracy and lower validation loss.

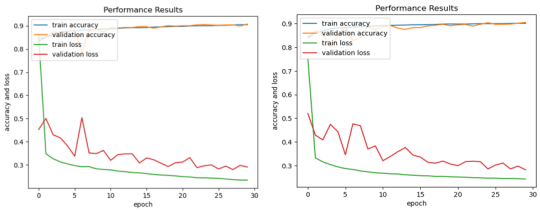

Figure 3.2: : Graph output of CNN model without data augmentation(left) and with data augmentation (right)

Looking at the output plots, we can see that the model with the data augmentation achieved a validation accuracy of 90.03% and a validation loss of 0.2936. The model without the data augmentation, on the other hand, achieved a validation accuracy of 89.69% and a validation loss of 0.3087. This indicates that the data augmentation technique was effective in improving the model's performance. It is worth noting that the training time of the model with the data augmentation is longer, which is expected because data augmentation generates additional images, increasing the number of training examples. However, the small extra training time is worth it because it leads to a better model performance. The model is optimised using binary cross-entropy loss functions and Adam optimiser. The binary cross-entropy function is a measurement of the dissimilarity between the predicted probability distribution and the true probability distribution. It sets up a binary classification problem between two classes for every class. The loss computed for every CNN output component is not affected by the other component values. We use cross entropy here as an ideal loss function, as it disciplines the model when a low probability is predicted for the target class.
Hyperparameters for a CNN include the number of filters, filter size, padding, pool size, dropout rate, activation function, number of neurons in the dense layers,the optimisers, batch and epoch size. These hyperparameters can be changed to improve the performance of the model. We chose to use the hyperparameters through trial and error throughout our IANNwTF course, as well as during the project. The pattern of gradually increasing the number of filters from 32 to 128 in deeper layers, as used in our CNN, was a suitable choice for balancing the model's complexity and capacity. The pattern allowed for the detection of the features while still trying to

avoid overfitting. The filter size of 3x3 for all convolutional layers is also a common choice for many image classification tasks, as it is the right size to capture both local and some global features. Padding is not only helpful in this case to reduce vanishing gradients but also to preserve spatial dimensions. Preserving the spatial dimensions of the input image in this CNN, by setting padding to the same size as the input images with "same", is important as the location of the cancerous cells can provide important diagnostic information. [19] Setting padding to "same" ensures that the output feature maps have the same dimensions as the input feature, helping to preserve the location information of the cancerous cell. Like the filter amount and size, the pool size is at a common setting. The 2x2 size for all MaxPooling2D layers helps reduce the spatial dimensions while preserving some spatial information. Bigger or smaller pool sizes were not necessary in our implementation. As already stated, the application of a dropout with the rate of 0.25 after each pooling layer has been demonstrated to enhance the performance of a deep learning model. There are two activation functions used in our CNN model. The first is the Rectified Linear Unit (ReLU) activation function, which is used for all convolutional layers. It is computationally efficient and helps to address the vanishing gradient problem that can occur. The other functions, such as tanh or sigmoid, were not reasonable for the CNN. The second activation function is used in the output layer and is the sigmoid activation function. As our breast cancer detection is a binary classification tasks, we deemed it reasonable to use the sigmoid, as it outputs a probability value between zero and 1, indicating to the two classes. In our code, there is one dense layer which has 128 neurons and one with 2 neurons. Having two neurons in the output dense layer corresponds to the two possible cases in the task at hand. Therefore, we knew we did not need to change this. The amount of neurons in the prior dense layer was chosen through trial and error, as too few neurons led to underfitting and too many to overfitting. The 128 neurons fit the complexity of the task and achieved good accuracy. For the optimiser, we chose the Adam optimiser with a learning rate of 0.0001, as Adam provided the best performance. Before increasing the batch size to 64, we tested the CNN on batch size 32, which we often used in our university course. However, we noticed the model **ask about it** Finally, we decided to let the CNN model run for 30 epochs to allow the model to modify its weights and biases thoroughly.

3.3 Transfer Learning Models

In our following implementation, we have used pre-trained models, which build up the concept of transfer learning. A model that has already undergone training on a specific dataset is known as a pre-trained model. The model stores the weights and biases that capture the features of that particular dataset. These learned features can be transferred to various other datasets. Pre-trained models provide several benefits. The models not only save time by leveraging the pre-existing knowledge of the model, but they also require minimal alteration for a new dataset as they already have been fine-tuned. [17] When using pre-trained models for transfer learning, it is crucial to choose the correct weights dataset. This is because the weights of a pre-trained model represent the learned features of the input data. Therefore, if the pre-trained model was trained on a dataset that is similar to the task at hand, the learned features will be more relevant and useful for the new task.

In our case we decide to use ImageNet weights dataset for all of the three pre-trained models, because is a large, diverse, and widely-used dataset, making it an ideal choice for our task.[11]

In addition, there are different components that can be altered. We can remove the default classifier and attach our own. Additionally, the input shape can be adjusted. The default input shape of a transfer learning model differs from each other but usually is around 224 x 224 for RGB images. However, the model can also be fed an input image other than the default. It is important to note, that the height and width must be more than 32 pixels and that the input shape can only be changed if the default classifier is excluded. For our model to have a different input shape, the breast histopathology images must be changed in the pre-processing as well.

3.3.1 VGG16

The first step in the our code using the technique transfer learning is to load the VGG16 model using Keras. The weight parameter is set to 'imagenet', which means that the pre-trained weights from the ImageNet competition will be used. As we want to add our own fully connected layers that are specific to our problem domain, we set the "Include_top" parameter to "False".

```
# create the InceptionV3 model
base_model = InceptionV3(weights='imagenet', include_top=False, input_shape=(75, 75, 3))

# add new classification layers on top of the base model
model = Sequential()
model.add(base_model)
model.add(Flatten())
model.add(Dense(512, activation='relu'))
```

Figure 3.3: :Snippet of the Inception V3 model with our own classifier and input image size

Next, we loop through all the layers in the VGG16 model and set their trainable attribute to False. This means that the weights for these layers will not be updated during training. By freezing the weights in the pre-trained layers, we can prevent them from being modified and potentially losing the valuable features they learned during the original training on the ImageNet dataset.

After the pre-trained model has been loaded and its layers have been frozen, we create a new Keras Sequential model, which will be used as the final model. We add the pre-trained VGG16 model to the new model using the add() method. We then add a Flatten() layer, which flattens the output of the VGG16 model into a 1D vector. This is necessary because the fully connected layers we add next require a 1D input.

We then add two fully connected layers to the model. The first layer has 64 units and uses the ReLU activation function, which is a common choice for neural networks. The second layer has 2 units and uses the sigmoid activation function, which is appropriate for binary classification problems. We use Dense() to add these layers to the model.

Finally, we import the Adam optimizer from Keras and use it to compile the model. The optimizer is responsible for updating the weights in the fully connected layers during training. We also specify the loss function and metrics to use during training.

Optimisation and Hyperparameters

The choice of learning rate and optimizer can have a significant impact on model performance, as they affect how the model updates its parameters during training. The batch size and number of epochs also play a role in determining the rate of convergence and potential overfitting of the model. These hyperparameters were chosen to balance the need for accurate results with the limitations of computational resources and training time.

Initially, we searched online for existing solutions or advice to decide on the appropriate hyperparameters for our model. Then, we used the "test and try" technique to fine-tune our model[21]. We decided to implement both Adam and RMSprop optimizers and started with the Adam optimizer, which has been frequently used with the VGG16 architecture. We observed a significant improvement in the model's performance with the Adam optimizer, but it took more time per epoch than the RMSprop optimizer.

Initially, we opted to use the Adam optimizer with a batch size of 32 and a learning rate of 0.001 since it has been commonly used with the VGG16 architecture . We noticed a noteworthy improvement in the model's performance after implementing this approach, achieving an accuracy of 0.9071 and taking approximately 53 seconds per epoch, which is considered good performance.

However, we were keen to explore further avenues for fine-tuning the model to improve its accuracy and efficiency. As a result, we decided to experiment with the RMSprop optimizer, which has also been found to be effective with VGG16 architecture.

To assess the performance of the RMSprop optimizer, we increased the batch size to 64 and maintained a learning rate of 0.001. The aim was to see if this would yield better results than the Adam optimizer. The larger batch size was expected to enable the optimizer to process more data at once, which could potentially lead to improved output.

After thoroughly evaluating the performance of both optimizers and their respective hyperparameters, we ultimately decided to proceed with the RMSprop optimizer and a batch size of 64 due to its superior accuracy and faster processing time.

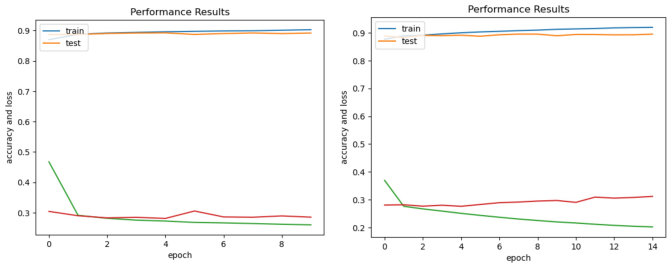

Figure 3.4: : Comparing the results from both optimizers with the RMSprop optimizer on the left

To summarize our final approach, we opted for a learning rate of 0.001 for the RMSprop optimizer, a commonly used starting value in deep learning. In terms of batch size, we chose 64, as it strikes a balance between maximizing training speed and fitting within available memory.

Initially, we set the number of epochs to 10, as this is generally considered a

good starting point for fine-tuning a pre-trained model. However, we increased it to 15 to allow the model to adjust its weights and biases for potentially improved accuracy and lower loss values. It is crucial to establish a solid baseline for future experiments, and choosing the right backbone and batch size are critical to achieving this.

Ultimately, selecting appropriate hyperparameters is vital for developing and fine-tuning a deep learning model for optimal performance. By carefully considering the available options and analyzing the results, we arrived at a configuration that we believe will enable us to achieve our objectives effectively.[13]

3.3.2 ResNet

We first loaded the ResNet50 model with the imagenet weights and adequate image size. In this case, the allocated image size of the original images, 50x50x3, was used. The default classifier is removed with the for loop and new classifier layers are added on top of the pre-trained model. We then, like by the other models, called the "Sequential ()" function to define a new model, which consists of the frozen ResNet50 model as the first layer. It was followed by a "GlobalAveragePooling2D()" layer, which averages the feature maps output by the ResNet50 model across spatial dimensions. Then we applied batch normalisation to normalise the activation's of the previous layer, followed by dropout regularisation to prevent overfitting. The following layer was a dense layer with 128 units and the ReLU activation function, followed by a batch normalisation and dropout regularisation. The process is repeated with another dense layer with 64 units and a final dense layer with two units and a sigmoid activation function. The final dense layer outputs the predicted probabilities for the two classes (benign or malignant). Whilst training and evaluating the performance of the model with "model.fit", we included two callback functions: "Early Stopping" and "ReduceLROnPlateau". They are used to monitor the training performance of the model and adjust the learning rate of the optimiser.

Optimisation and Hyperparameters

For our ResNet50 model, certain hyperparameters were specified, including a learning rate of 0.0001 for the Adam optimizer, a batch size of 64 and a total of 10 epochs. The learning rate was set to 0.001 as it was a good beginning. Initially, we had a ResNet50 model that has a very high accuracy rate and low loss rate at the first epoch. Although, we researched into it and found information to back the concept up, we decided to try and change the parameters. Some research papers such as "Transfer Learning with ResNet-50 for Malaria Cell-Image Classification" conducted by A. Sai Bharadwaj Reddy and D. Sujitha Juliet had a similar output to ours.[1], the majority of research papers did not. After discussing it with each other and a tutor, we came to the conclusion that the model does not resemble actual learning. We decided to change hyperparameters and try regularisation techniques. As we had already implemented techniques for the overfitting without a success in the high accuracy rate problem, we decided to look into the hyperparameters. Once again, the trial and error technique lead us to a solution. The learning rate at 0.001 was too high for the model.

We initially had a smaller batch size of 16 to test the ResNet. However, we decided to increase it to 64 to create a more stable an accurate estimation, as well

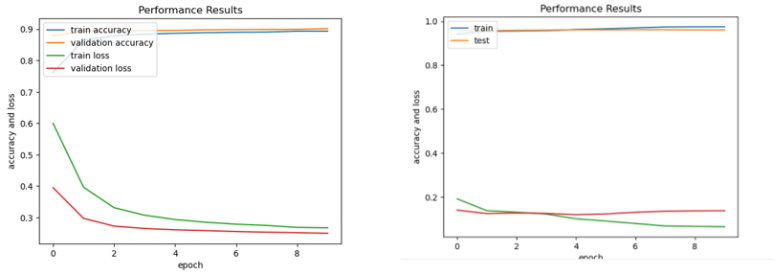

Figure 3.5: : Graph output of ResNet50 model with learning rate 0.001 (right) and 0.0001 (left)

as creating a similarity to the other pre-trained model implementations.

Similarly, to the other models, we implemented a ReLU activation function for the ResNet50's dense layers. Using ReLU activation in the dense layers of the classifier allows the network to learn non-linear relationships between the features extracted by the convolutional layers and the target class, which helps to improve the model's accuracy. In addition, we imported an Adam optimiser from Keras with a learning rate of 0.0001. Originally, the model had a slight overfitting. This we fixed with a dropout layer, where a fraction of the inputs are randomly ignored.

During our implementation, the system of Kaggle often was restarted due to storage issues. Therefore, we implemented a reduction of the sample sizes to allocate the memory problems in Kaggle. By reducing the images, the issue was resolved and we were able to effectively work with the model.

```
#Reduce Sample Size for allocating memory Problems in kaggle

X_train = X_train[0:50000]
Y_train = Y_train[0:50000]
X_test = X_test[0:30000]
Y_test = Y_test[0:30000]
```

3.3.3 InceptionV3

Before we can train our model, we need to preprocess our images to make sure they are in a format that our model can use. In our case, we will be resizing all of our images to 75x75 pixels. This is because we read that this is the smallest size that the InceptionV3 model works with. [16]

In the preprocessing stage, the same steps as before are performed with the exception of resizing the images to 75x75 pixels using OpenCV's resize() function. This step ensures that all images have the same dimensions, which is essential for the model to function correctly.

After that is done we created a convolutional neural network (CNN) using the InceptionV3 model for image classification. The InceptionV3 model is pre-trained on the ImageNet dataset and is used as a base model in this code. The top layers of the model are removed, and new classification layers are added to the top of the model. The new layers consist of a Flatten layer, a Dense layer with 512 units and ReLU activation, a Dropout layer with a rate of 0.5, and a Dense layer with 2 units

and softmax activation, which is the output layer. The base model's convolutional layers are frozen to avoeid overfitting, and only th newly added layers are trained. The model is compiled with an optimizer of Rmsprop, a loss function of binary cross-entropy, and accuracy as the evaluation metric. The fit method is called to train the model on training data and validate on testing data. The training process is run for 15 epochs with a batch size of 32. The training results are then printed out as the loss and accuracy of both training and validation data at each epoch.

```
# create the InceptionV3 model
base_model = InceptionV3(weights='imagenet', include_top=False, input_shape=(75, 75, 3))

# freeze the convolutional layers in the base model
for layer in base_model.layers:
    layer.trainable = False


from keras.callbacks import EarlyStopping

# define early stopping criteria
early_stop = EarlyStopping(monitor='val_loss', patience=10, verbose=1, mode='min')


# add new classification layers on top of the base model
model = Sequential()
model.add(base_model)
model.add(Flatten())
model.add(Dense(512, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(2, activation='softmax'))
```

Additionally, to prevent overfitting and unnecessary training epochs, we implement the early stopping technique using the EarlyStopping function. The EarlyStopping function monitors the validation loss and has a patience of 10. This technique helps to improve the model's generalization ability by stopping the training process when the model's performance on the validation set starts to degrade. In other words, early stopping ensures that the model does not learn the noise in the training data and achieves better performance on unseen data.

Optimisation and Hyperparameters

After running the model for the first time, we noticed that the model is likely overfitting. This observation was made by seeing that the validation accuracy is stuck around 84%. To tackle this issue, we implemented an early stopping technique,as stated before, using the EarlyStopping function with a patience of 10 and monitored the validation loss. However, we didn't see any improvement in the validation accuracy.

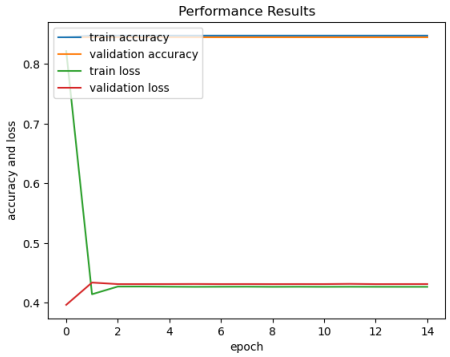

Figure 3.6: : Graph representing the overfitting of the InceptionV3 model

We opted to investigate if the issue resided in the processing of the data. To achieve this, we chose to visualize a selection of resized data images, with the intent of identifying any potential issues. The resulting images can be observed below.

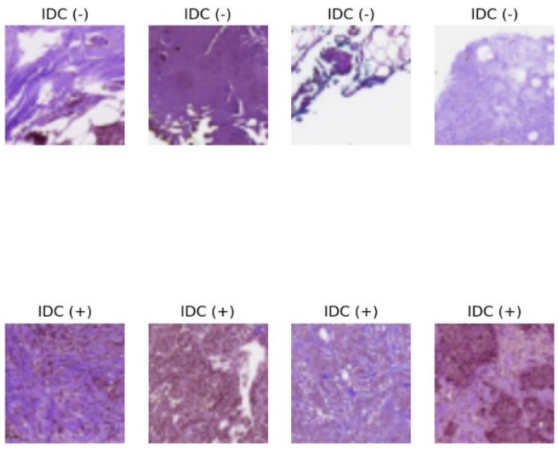

Figure 3.7: : Resized images

As one can see, the images appear flawless and suitable for use. As a result, we eliminated this possibility from our list of potential causes.

After determining that the problem did not stem from data processing, our attention turned to the model itself. Specifically, we began to explore various techniques and strategies for addressing its overfitting issue.

Our next step was to implement a dropout layer of 0.5. This technique can help to prevent overfitting by reducing the dependence of the model on any individual neuron. After adding this layer, we retrained the model and saw a slight improvement in the validation accuracy.

We also looked into the hyperparameters we used. At first, we used the Adam optimizer, which is a popular optimizer for deep learning. However, after researching other people's work, we saw that InceptionV3 typically performs better with RMSprop optimizer [15] We then changed the optimizer to RMSprop and retrained the model. While there was a slight improvement in accuracy, the accuracy was still not increasing beyond a certain point.

To further optimize the model, we tried adjusting other hyperparameters such as the learning rate, batch size, and the number of units in the dense layer. We also experimented with adding more layers to the model, but ultimately we did not see any significant improvement in the validation accuracy beyond 84%.

Chapter 4

Result

4.1 CNN result

Our first implemented model was the CNN. Let's take a look at the output of the model. Initially, the model has a loss of around 75.35% with an accuracy of 83.88% on the training data, which highlights a sub-optimal level of accuracy. Simultaneously, the model achieves a validation accuracy of 84.48% and a loss of 52%. In the subsequent epochs, the model's accuracy improved gradually, with the accuracy reaching 88.14% on training data and its validation accuracy reaching 84.76% already in the 5th epoch. By the 20th epoch, the model has an accuracy of 89.82% on the training data and 89.10% on the validation data. This means the model is able to identify breast cancer with an accuracy of almost 90%, which is a significant improvement from its initial accuracy circulating 84%. The loss, simultaneously, decreased throughout training, reaching 0.2552 for training data and 0.3064 for the validation data at the 20th epoch. It is important to the note, that there were fluctuations in the accuracy and loss metrics. For instance, certain epochs demonstrated a decline in accuracy, particularly in the initial stages where the accuracy levelled off at 80%. Additionally, there were instances where the loss metric increased, as evidenced in the 6th and 7th epoch, with the loss value spiking to 0.4762 from its prior value in the low thirties. However, overall, the model showed consistent improvement in accuracy over the course of the training reaching a training accuracy of 90.07% and a validation accuracy of 90.47%. To visualise our CNN model's progress,

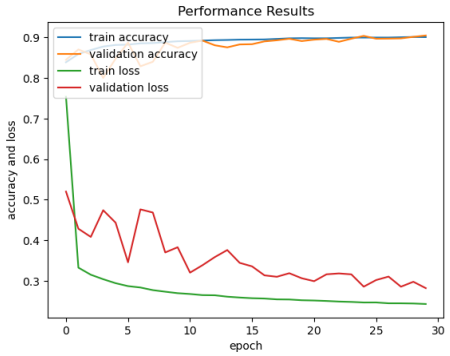

Figure 4.1: : Graph representing the results of the CNN

we implemented a basic line graph with both the loss and accuracy of the training

and test data. The graph visualised both depicts the growth of the accuracy and decrease of loss, as well as highlighting the fluctuations the model experienced at the beginning. Overall, the graph reinforces the conclusion that the CNN model was able to learn from the training set, improve its performance and classify the samples pretty accurately. We also decided to visualise the output of the CNN model with further graphs. The first graph is the Confusion Matrix.

True Positive (TP)	False Negative (NP)
False Positive (FP)	True Negative (TN)

: Basic representation of a confusion matrix

This is a Confusion Matrix, which is a powerful tool for evaluating the performance of a classification model. It helps to visualize the performance of the model by showing the number of correct and incorrect predictions in each class. The rows of the matrix represent the true labels, while the columns represent the predicted labels. Each cell of the matrix shows the number of data points that belong to a particular combination of true and predicted labels. The colour of each cell indicates the density of the data points. A confusion matrix can help to identify which classes the model is good at predicting and which ones it is not. With the model, one can calculate further statistical measurements to better understand the model's accuracy. One can calculate the overall accuracy, the sensitivity or recall, specificity and precision.

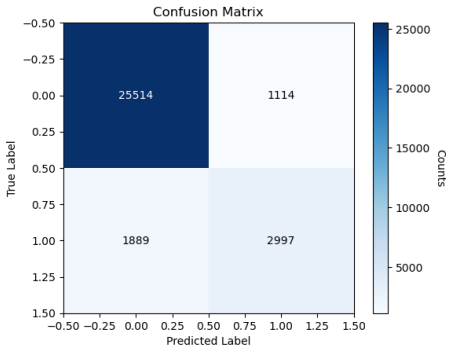

Figure 4.2: : Confusion Matrix of the CNN model

In the case of our CNN, the confusion matrix indicates that out of all the samples, the model accurately predicted 25641 as true positive and missed 987 as false negatives. In addition, the CNN made 1940 false positive predictions and 2946 true negatives. From this data, we calculated an accuracy of 90.7%. In addition, both the Recall of 96.3% and precision of 93% calculated, highlighted that the model correctly identifies a high number of cancer-positive images which links to the overall performance improvement. The fluctuation of the accuracy during training is also highlighted by the calculations, more specifically the specificity measurement, which shows the performance in predicting negative outcomes. The model achieved a specificity of 60.3%. The confusion matrix and its calculations reinforce the improving CNN performance with a coupe of fluctuations.
The second visualisation is the ROC curve.

Receiver operatin characteristics (ROC) graphs are useful for organising classifiers

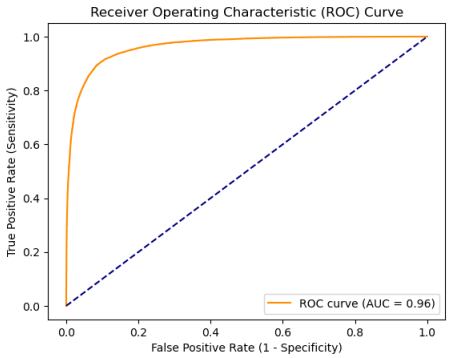

Figure 4.3: :Receiver Operating Characteristic (ROC) Curve for CNN Model

and visualising their performance. In recent years, there has been an increase in the use of ROC graphs in machine learning. Prior to its popularity growth, the graphs were commonly used in medical decision making. We, therefore, found it fitting to implement the graph type for our models.[7] ROC graphs are two-dimensional graphs in which TP rate is plotted on the Y axis and the FP rate is plotted on the X axis, depicting the relative trade-offs between benefits (TP) and costs (FP). The area under the curve is a summary statistic that measures the overall performance of the model across all possible thresholds. It is a metric used to evaluate the performance of a binary classification model. A single number is provided that summarises the overall performance of the model. The area under the curve ranges between zero and one, with a value of 1 indicating a perfect classification and a value of 0.5 indicating random guessing. The ROC curve of our CNN model highlights the high level of accuracy in the model's predictions. This is not only depicted by the curve being very close to the top left corner, which is a big indication, but also by the calculated AUC. The AUC value of 0.96 for the model implies an almost perfect accuracy classification.

The final graph, we decided to visualise was a histogram of the prediction errors. As the model had a high specificity, we thought it would be reasonable to visualise not only the great accuracy but also the errors. A prediction error graph in deep learning models plot the distribution of the difference between the predicted values and the actual values. It helps to visualise the spread and direction of the errors, and to identify any patterns or trends in the errors that may suggest areas for improvement. A perfect model would have all the errors at 0, but in reality, some deviation is expected. The goal is to minimize the deviation as much as possible. The graph is typically plotted as a scatter plot. We have implemented the distribution of these predictions as a histogram for a better representation. [12]

As we can see above, the prediction errors of the CNN model are in the majority surrounded by the zero, with a few cases spread around. This highlights, once again, that the majority of images are predicted correctly with a couple of incorrect classifications.

In summary, the output indicates that the CNN model was able to detect breast cancer with a high degree of accuracy. All of output visualisation help to understand the accuracy and loss that the model created. The model's performance was consistent across both the training and validation sets, suggesting its generalisability to new data.

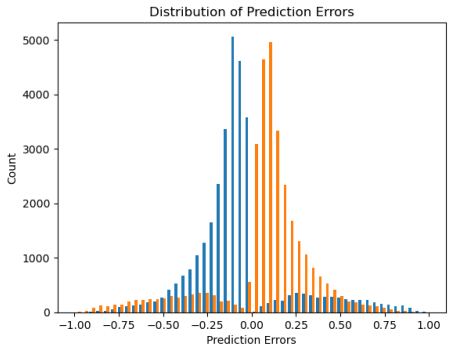

Figure 4.4: : Histogram of prediction error for CNN model

4.2 VGG16 results

The initial training of the VGG16 model was a resounding success, producing outstanding results. The transfer learning technique was utilized, and as expected, it performed exceptionally well with high accuracy right from the start. Moreover, the model demonstrated impressive learning progress, achieving remarkable performance in a relatively short period of time.

Furthermore, one of the most striking aspects of the VGG16 model was its exceptional speed. The model was able to process data at a remarkable pace, making it a highly efficient tool for machine learning tasks. This impressive speed was particularly noteworthy given the complexity of the model, and it speaks to the efficacy of its architecture and design.

```
Epoch 1/10
1232/1232 [==============================] - 34s 22ms/step - loss: 0.4672 - accuracy: 0.8698 - val_loss: 0.3039 - val_accuracy: 0.8869
Epoch 2/10
1232/1232 [==============================] - 26s 21ms/step - loss: 0.2911 - accuracy: 0.8875 - val_loss: 0.2896 - val_accuracy: 0.8869
Epoch 3/10
1232/1232 [==============================] - 22s 18ms/step - loss: 0.2811 - accuracy: 0.8918 - val_loss: 0.2826 - val_accuracy: 0.8902
Epoch 4/10
1232/1232 [==============================] - 22s 18ms/step - loss: 0.2751 - accuracy: 0.8939 - val_loss: 0.2844 - val_accuracy: 0.8917
Epoch 5/10
1232/1232 [==============================] - 26s 21ms/step - loss: 0.2721 - accuracy: 0.8958 - val_loss: 0.2807 - val_accuracy: 0.8922
Epoch 6/10
1232/1232 [==============================] - 22s 18ms/step - loss: 0.2676 - accuracy: 0.8972 - val_loss: 0.3052 - val_accuracy: 0.8871
Epoch 7/10
1232/1232 [==============================] - 22s 18ms/step - loss: 0.2660 - accuracy: 0.8986 - val_loss: 0.2856 - val_accuracy: 0.8900
Epoch 8/10
1232/1232 [==============================] - 26s 21ms/step - loss: 0.2638 - accuracy: 0.8992 - val_loss: 0.2846 - val_accuracy: 0.8919
Epoch 9/10
1232/1232 [==============================] - 26s 21ms/step - loss: 0.2615 - accuracy: 0.9010 - val_loss: 0.2890 - val_accuracy: 0.8900
Epoch 10/10
1232/1232 [==============================] - 26s 21ms/step - loss: 0.2596 - accuracy: 0.9027 - val_loss: 0.2849 - val_accuracy: 0.8917
```

Figure 4.5: : Neural network training results: 90% accuracy achieved in 10 epochs with an average speed of 23 seconds per epoch

The results indicate that the model's loss decreased from 0.4672 to 0.2596 during the 10 epochs, indicating that the model was able to learn from the training set and improve its performance. The accuracy increased from 0.8698 to 0.9027, which indicates that the model was able to classify the samples with a high degree of accuracy.The training took around 220 seconds, with each epoch taking an average of 22-26 seconds.This indicates that the model is able to process data efficiently and quickly, which is a really desirable characteristic.

We can also take a look at the graph we plotted to visualize the results better. When we examine the plotted results, we can see that the accuracy steadily increases while the loss consistently decreases over the 10 epochs of training. This is a clear indication that the VGG16 model is efficiently learning and improving its

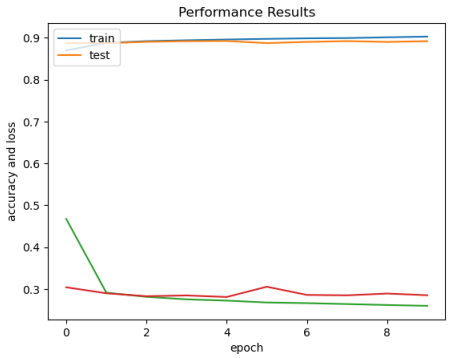

Figure 4.6: : Graph representing the results of the VGG16 model

performance. Additionally, the small difference between the training and testing accuracy and loss values suggests that the model is not overfitting to the training data. Therefore, the performance results depicted in the graph reinforce the conclusion that the VGG16 model is performing exceptionally well.

Additionally, we evaluated the VGG16 model's performance by generating a confusion matrix, which revealed some useful insights. Specifically, the confusion matrix shows that out of a total of 26,614 test images, the model correctly predicted 25,948 as true positives and 2,156 as true negatives. However, there were 680 instances where the model falsely predicted a negative result for an actual positive case (i.e., false negatives) and 2,730 instances where it falsely predicted a positive result for an actual negative case (i.e., false positives).

True Positive = 25948	False Negative = 680
False Positive = 2730	True Negative = 2156

VGG16 Confusion Matrix

‘ Furthermore, these results indicate that the VGG16 model performed relatively well, correctly identifying the majority of positive samples with a high true positive value. However, there were still some instances where positive samples were incorrectly predicted as negative (FN), and a relatively large number of negative samples were incorrectly classified as positive (FP). These insights can help to inform further adjustments to the model to improve its performance.

After seeing the distribution of the True positive and False positive cases we decide to plot a Receiver Operating Characteristic (ROC) curve. This curve is a graphical plot that displays the performance of a binary classifier system. It shows the relationship between true positive rate (TPR) and false positive rate (FPR) across a range of threshold values. A perfect classifier will have a ROC curve that passes through the top-left corner of the plot.

The ROC curve generated for the VGG16 model shows a high level of performance, with an AUC (area under the curve) value of 0.96. The curve closely hugs the top-left corner of the plot, indicating that the model has a very high true positive rate (sensitivity) and a very low false positive rate (specificity) for all possible threshold values. This indicates that the model is able to accurately distinguish between positive and negative samples, making it a valuable asset for any data science project.

Overall, the excellent results of the VGG16 model's initial training highlight the potential of transfer learning in machine learning. With its impressive accuracy,

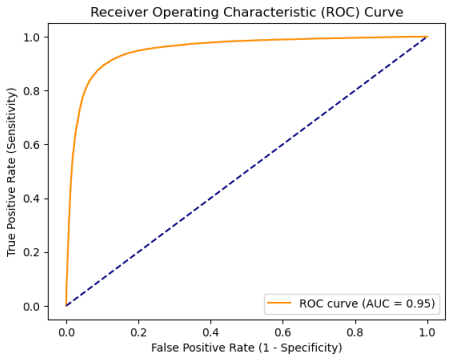

Figure 4.7: :Receiver Operating Characteristic (ROC) Curve for VGG16 Model

speedy learning progress, and efficient processing, the VGG16 model is a valuable asset for data science projects.

4.3 ResNet results

The output provided for the ResNet50 shows remarkable progress with each epoch having a better performance than its prior epoch. We ran the model for ten epochs initially, as this was the amount that our system could safely implement for the ResNet. During the first epoch, the model demonstrates a training accuracy of 76.21% and a loss of 0.5999. At the same time, the validation accuracy reached 87.95% with a validation loss of 0.3951. The model continued to train and learn from its errors, with each epoch demonstrating even better performance. By the final epoch, the model showcased a training accuracy of 89.34% and a loss of only 0.2498. The validation accuracy was also noteworthy at 90.15% with a validation loss of 0.2498.

Like all are other models, we decided to visualise the ResNet50 with different types of plotting styles.

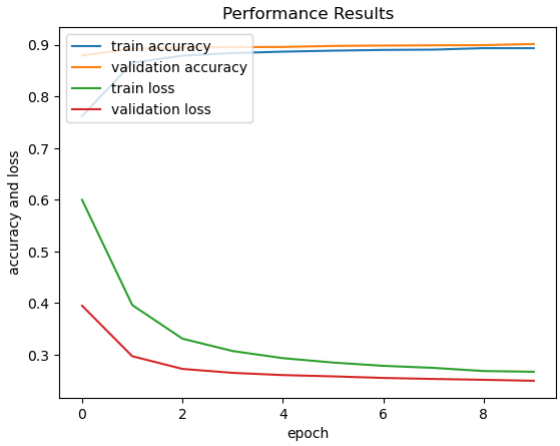

Figure 4.8: :Graph representing the results of the ResNet50 model

The first graph we plotted for this model, showcased the beginning accuracy in both validation and training. To be able to have a closer look at the loss and

accuracy output, we decided to plot them individually as well. This showed us in more detail the sudden increase in accuracy and sudden decrease in loss. The confusion matrix of the ResNet indicated that out of the total of samples, there were 25852 true positives with 2295 instances that were incorrectly classified as positive, resulting in a false positive count. This means that a big majority of the positive samples were classified correctly. There were 810 instances of incorrectly classified samples as negative, false negatives. Finally, the true negative count is 2557, which highlights the number of correct identifications for negative cases. Looking at these numbers, it seems like the model is doing well in terms of correctly identifying positive cases. However, the false positive is also relatively high, indicating that there are cases where the model is incorrectly classifying negative instances as positive. Through the further calculations done with the confusion matrix measurements, we computed an accuracy rate of 90.15%. To be expected the recall rate was pretty high with 96.92%, whilst the specificity rate was pretty low with 49.65%. The precision rate approximated at 91.89%. Overall, these calculations align with the evaluation of the output information. The other two plots both depicted the high accuracy performance of the model as well. The ROC curve was set very close to the top left corner with an AUC value of 0.96.

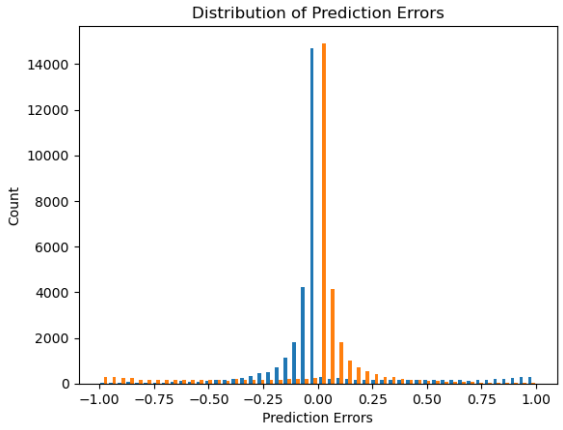

Figure 4.9: : Prediction error graph representing the results of the ResNet50 model

The prediction error graph exhibits a significantly higher magnitude of error at the zero rate as compared to all other values, which highlights the good accuracy very well. Overall, the graphs depict the interesting output of the ResNet50 very well.

4.4 InceptionV3 results

The InceptionV3 model, which was implemented using the transfer learning technique, displayed remarkable results in terms of accuracy right from the beginning. During the training pr ocess, the model achieved a training accuracy of 0.8386 and a validation accuracy of 0.8453, which is remarkable considering the complexity of the task.

However, as the training progressed, the accuracy remained stagnant around 85%, indicating a limitation of the model. Which you can notice on the plot above.

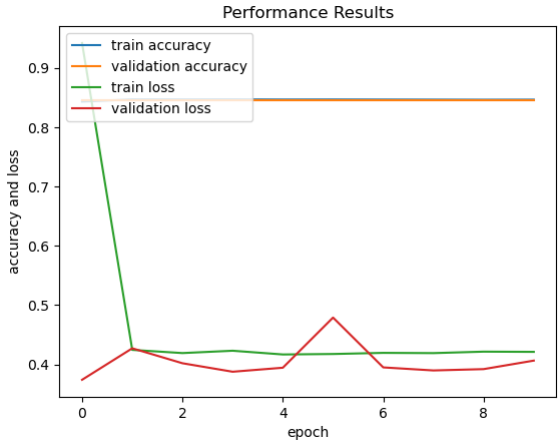

Figure 4.10: :Plot visualizing the accuracy and the loss during training

Despite this, the loss function during training showed a significant reduction, indicating that the model was learning effectively.

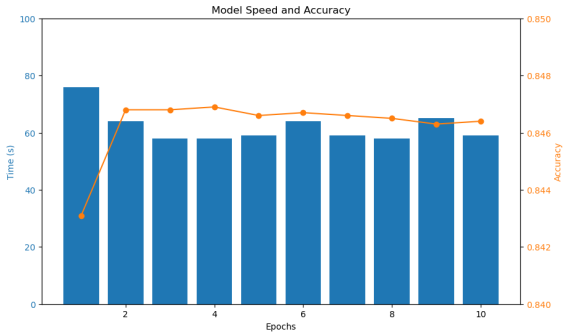

Figure 4.11: :Bar chart depicting the accuracy and the speed of the model

One of the drawbacks of the InceptionV3 model,that is depicted on figure 4.9, is that it is relatively time-consuming to train, taking approximately 62 seconds per epoch. This is slower compared to other models such as VGG16 and ResNet.However, the model exhibited high precision in image classification, indicating that it is a powerful tool for this task.

It is important to note that there is a possibility that we could not overcome the problems with the model or our data, and this could limit the accuracy of the model. Therefore, it is crucial to thoroughly examine the data and the model's performance to determine the model's effectiveness.

In conclusion, the InceptionV3 model with transfer learning technique showed remarkable results in terms of accuracy and demonstrated its capability for image classification. However, its limitations include its slow training time and its stagnant accuracy in subsequent epochs. Further fine-tuning of the model could improve its efficiency and accuracy, but this would require significant computational resources and time.

4.5 Comparison between all of the approaches

Based on the evaluation of the four approaches for image classification using deep learning, it was found that the use of a basic convolutional neural network (CNN) and pre-trained VGG16 model achieved the highest accuracy at 0.902 and slightly behind that the ResNet50 with an accuracy of 0.901, while transfer learning with InceptionV3 achieved slightly lower accuracy of 0.84.

In terms of training time, the basic CNN took the longest time to train, at 82 seconds per epoch, followed by InceptionV3 at 58 seconds per epoch. On the other hand, the pre-trained VGG16 model had the fastest training time, at 34 seconds per epoch, however the ResNet also showed impresive training time at 36 second pre epoch on average.

While the basic CNN achieved high accuracy, it required more computational resources and longer training time. The VGG16 model performed well in terms of accuracy and training time, likely due to its deeper architecture and pre-trained weights.

The InceptionV3 model with transfer learning technique yielded impressive results in terms of accuracy from the start, with a training accuracy of 0.8386 and a validation accuracy of 0.8453. However, as mentioned before, the accuracy remained stagnant around 85% in subsequent epochs, which was a limitation of the model. The loss function during training also reduced considerably, indicating that the model was learning well. Despite the stagnation in accuracy, the model was able to classify images with a high degree of accuracy. However, the training of the model was time-consuming, taking about 58 seconds per epoch, which was slower than other models.

The ResNet model performed better than the Inception V3 model in terms of accuracy, with a validation accuracy of 0.901. Additionally, it was able to learn quickly, achieving its highest accuracy within the first few epochs of training.

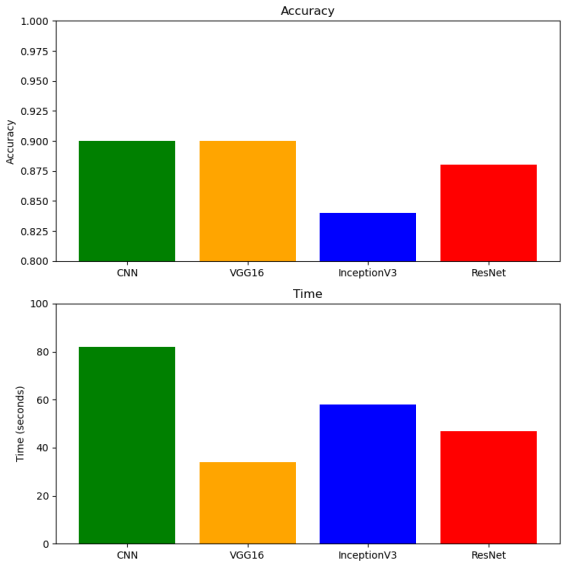

Figure 4.12: :Bar chart visualizing and comparing the four approaches

In conclusion, the choice of model for image classification depends on the specific requirements of the task, such as desired accuracy, training time, and computational

resources. The pre-trained VGG16 and the ResNet50 models are good choices for tasks that require high accuracy and fast training time, while the basic CNN can achieve high accuracy at the expense of longer training time. Transfer learning with InceptionV3 can also achieve high accuracy, but requires more computational resources and longer training time.

Chapter 5

Literature review

5.1 Different approaches

In recent years, deep learning techniques have shown promising results in detecting and classifying breast cancer from mammography images. To inform our approach, we reviewed existing literature on similar studies. Two papers that particularly stood out were "Transfer Learning and Fine Tuning in Breast Mammogram Abnormalities Classification on CBIS-DDSM Database" by Falconi et al.[9] and "Breast Cancer Detection in Mammograms using Deep Learning Techniques" by Jaamour[2]. All studies show that deep learning techniques have the potential to improve the accuracy and efficiency of breast cancer detection and classification. In our project, we will build on these findings by comparing the performance of transfer learning models to a basic CNN architecture and other approaches for detecting breast cancer.

The purpose of the first paper "Transfer Learning and Fine Tuning in Breast Mammogram Abnormalities Classification on CBIS-DDSM Database" by Falconi et al.[9] is to classify mammogram abnormalities using Transfer Learning with various ConvNets. The authors use ImageNet pre-trained ConvNet models and a Fine-Tuning technique to address the classification problem of mammogram abnormalities in this paper. The study's goal is to classify region of interest images from the CBIS-DDSM dataset's mass tumors, and the results show that Fine Tuning can train an accurate classifier while overcoming overfitting. When there are few samples available, such as in mammogram datasets, Transfer Learning (TL) and Fine-Tuning (FT) are discussed as methods for training deep ConvNets.[9] Fine-tuning entails re-training some of the pre-trained ConvNet's final layers with new images to improve its performance on a target learning task. Fine-tuning, as opposed to transfer learning, entails selecting a set of layers to be re-trained as well as several layers to be preserved or frozen. Transfer learning necessitates more computing resources and training time. The authors also discuss the common machine learning problem: overfitting. It occurs when a model is overly complex and has a high variance, which can be addressed with techniques such as Data Augmentation, Regularization (e.g., L2 regularization and Dropout), and Early Stopping. Overparameterization is common in ConvNet models. Because they retrain fewer parameters than models with randomly initialized weights, TL and FT can help prevent overfitting. In their experiment, the authors used Dropout and Early Stopping as regularization techniques, as well as data augmentation.[9] To receive a wide overview on the effects

of Transfer Learning and Fine-Tuning the authors worked with 20 different models. Including VGG16/19, ResNet50/101/152, DenseNet121/169/201 and InceptionV3. We will only list the results for the best performing model and the models that were also used in our project (ResNet, InceptionV3 and VGG16). The performances in the paper were measured with the area under the ROC curve(AUC), F1 Score(F1), the overall accuracy(ACC). In addition to that the overfitting was calculated by comparing the train accuracy (train-acc) and the validation accuracy (valid-acc) and this ratio was defined as ß.

$$\text{ß} = \text{train-acc/valid-acc}$$
$$\text{ACC} = \text{TP} + \text{TN/TP} + \text{TN} + \text{FP} + \text{TN}$$
$$\text{F}_1 = 2TP/2TP + FP + FN$$
$$\text{AUC} = \tfrac{1}{2} * (1 + \text{TPR - FPR})$$

Model	AUC	$\text{F}_1 Score$	ACC
ResNet-50-TL	0.861	0.858	0.861
Inception-v3-TL	0.778	0.781	0.779
VGG16-TL	0.644	0.654	0.644

the TL stands for Transfer Learning

In regards to the Transfer Learning the ResNet-50 was overall complared to the 20 models the best performing one while the VGG16 performed the worst.[9]

Model	train-acc	test-acc	ß
VGG16-TL	0.65	0.64	1.01
ResNet-50-TL	1.00	0.86	1.16
Inception-v3-TL	0.99	0.78	1.27

the TL stands for Transfer Learning

But after calculating the overfitting ratio ß the authors pointed out that the VGG16 had almost no overfiting while the InceptionV3 had a high overfitting and the ResNet50 a medium one. As already mentioned the authors also experimented with the concept of Fine-Tuning (FT), where the results differ alot from the Transfer Learning (TL).[9]

Model	AUC	$\text{F}_1 Score$	ACC
VGG16-8-FT	0.844	0.85	0.844
ResNet-50-160-FT	0.5	0	0.5

the FT stands for Fine-Tuning

The authors did not perform Fine-Tuning on the InceptionV3 and the only relevant numbers for our project in this case are the ResNet50 and the VGG16. Looking at the results it is interesting that the accuracy of the VGG16 model is far better than the ResNet50.[9] In conclusion the VGG16 achieves the best results for the dataset CBIS-DDSM (ACC = 0.84, ß = 1.07, F1 = 0.85) when suing the Fine-Tuning method. But when using the Transfer Learning method the ResNet50 performs the best with ACC = 0.86, ß = 1.16 and F1 = 0.86.

The second paper "Breast Cancer Detection in Mammograms using Deep Learning Techniques" by Jaamour[2] adresses the same problem as the other papers and also works on the CBIS-DDSM Database that consists of 2,620 scanned film mammography studies with the ConvNet models VGG19, ResNet50, InceptionV3, DenseNet121 and MobileNetV2. The author performed a similiar study like Falconi et al.[9] but in addition to that he also worked with the mini-MIAS Dataset that consists of only 322 scans and performed the same Deep Learning Techniques on both datasets. To find the best accuracy the author on the one hand varied with the amount of data augmentation on the mini-MIAS dataset. For the CBIS-DDSM dataset on the other hand the author performed variations in the class weights, the size of the input image and the amount of applied Transfer Learning.[2] Without any form of data augmentation the author gathered following results for the CBIS-DDSM dataset:

Model	$F_1 Score$	ACC
VGG19	0.638	0.639
ResNet50	0.612	0.61
InceptionV3	0.626	0.627
MobileNetV2	0.663	0.664

This table shows that the MobileNetV2 performes better than any other model with the highest accuracy and F1 score. And the ResNet50 performes the worst out of all models. [2] The author then performed different amounts of data augmentation (none, balanced and double) on the mini-MIAS dataset and came to the following results. It is important to mention that he only worked with the VGG19 and the MobileNetV2 architectures:

Model	Data Augmentation	$F_1 Score$	ACC
VGG19	None	0.507	0.646
VGG19	Balanced	0.404	0.369
VGG19	Double	0.507	0.646
MobileNetV2	None	0.507	0.646
MobileNetV2	Balanced	0.409	0.415
MobileNetV2	Double	0.507	0.646

This table the author showed that the data augmentation is very inefficient when working with a small dataset like the mini-MIAS.[2] After test on the mini-MIAS dataset the author concluded that data augmentation is very inefficient when working with a small dataset.

Coming back to the CBIS-DDSM dataset, the author concluded the following results after performing variations in the class weights, the size of the input image and the amount of applied Transfer Learning. Here is again important to mention that the author performed these experiments only on the MobileNetV2 architecture because this model performed the best out of all the others. The most positive result of an accuracy of. [2] "This deep learning pipeline exposed the effects of the different techniques used. The most positive result (0.6708) on CBIS-DDSM came from transfer learning techniques, using ImageNet weights with MobileNetV2 and binary mini-MIAS weights for the custom MLP layers, coupled with class weight techniques for balancing the dataset."[2].

Model	$F_1 Score$	ACC
MobileNetV2	0.6648	0.6708

5.2 Comparison

In this study, we aimed to compare the results of our neural network architecture with those reported in two previous studies by Jaamour [2] and Falconi [9]. Both studies used the CBIS-DDSM dataset, which comprises 2,620 scanned film mammography studies, while we used a much larger dataset of 277,524 patches of breast histopathology images from Paul Mooney on Kaggle.

One issue we encountered was class imbalance, as our dataset consisted of more than two times as many negative IDC (invasive ductal carcinoma) images than positive IDC images. This imbalance can lead to performance differences in the models. Therefore, the comparison of the different architectures and their accuracy is influenced by the use of a larger dataset with class imbalance.

Despite this limitation, it is important to highlight the differences and similarities in these studies to enhance our understanding of the effectiveness of transfer learning. Notably, Jaamour and Falconi came to different conclusions regarding the most effective neural network architecture for the CBIS-DDSM dataset.

Overall, our study aims to contribute to the growing body of literature on the use of deep learning models for breast cancer detection, particularly regarding the impact of dataset size and class imbalance on model performance.

Author	Model	AUC	$F_1 Score$	ACC
Our Result	ResNet50	0.96	0.943	0.901
Falconi	ResNet50	0.844	0.858	0.861
Jaamour	ResNet50	/	0.612	0.61
Jaamour	MobileNetV2	/	0.6648	0.6708

Author	Model	AUC	$F_1 Score$	ACC
Our Result	VGG16	0.96	0.938	0.903
Falconi	VGG16	0.644	0.654	0.644
Jaamour	VGG19	/	0.638	0.639

Author	Model	AUC	$F_1 Score$	ACC
Our Result	InceptionV3	/	/	0.845
Falconi	InceptionV3	0.778	0.781	0.779
Jaamour	InceptionV3	/	0.626	0.627

The studies conducted by Falconi et al. [9] and Jaamour[2] investigated the effectiveness of various neural network architectures on the CBIS-DDSM dataset. While Falconi concluded that ResNet50 is the most promising architecture for this dataset, Jaamour reported that MobileNetV2 outperforms all other architectures, including ResNet50, VGG19, and InceptionV3.

One potential explanation for the observed differences in results is the variation in pre-processing techniques employed by the authors. Jaamour utilized whole images for their analysis, while Falconi cropped the images around Regions of Interest (ROIs). ROIs refer to sub-regions within a picture frame that capture the object of

interest, and this form of segmentation is often used to reduce noise in the data and improve the overall model performance.[9]
The use of different pre-processing techniques may have influenced the results reported by Falconi and Jaamour. Therefore, it is important to consider the impact of pre-processing techniques on the accuracy of neural network architectures when conducting image classification tasks.[2] ,[9]

The purpose of this study is to compare the accuracy of various neural network architectures on a specific dataset. The three tables present the results of the different authors and their/ our models in terms of AUC, F1 score and accuracy (ACC) for classifying IDC in breast histopathology images. The first table compares our results to those of Falconi and Jaamour, who used ResNet50 and MobileNetV2. Our ResNet50 achieved the highest accuracy of 0.901, followed by Falconi's ResNet50 (0.861) and Jaamour's MobileNetV2 (0.6708). [2] ,[9]
The second table compares our VGG16 results with those of Falconi's VGG16 and Jaamour's VGG19. Our VGG16 achieved the highest accuracy of 0.903, followed by Falconi's VGG16 (0.644) and Jaamour's VGG19 (0.639). [2] ,[9]
The third table compares our InceptionV3 results with those of Falconi's InceptionV3 and Jaamour's InceptionV3. Our InceptionV3 achieved an accuracy of 0.845, while Falconi's InceptionV3 achieved an accuracy of 0.779 and Jaamour's InceptionV3 achieved an accuracy of 0.627.[2] ,[9]
The observed differences in accuracy may be attributed to various factors, including the choice of dataset. In this study, a larger dataset with a more extensive collection of training and test data was employed. This could have led to a different overall accuracy. Additionally, the pre-processed images used in this study had a size of 50x50, whereas Falconi used ROI images of size 320x320, and Jaamour used images of size 224x224.[2] ,[9]
Overall, our results achieved the highest accuracy across all models compared to those of Falconi and Jaamour. However, the differences in the dataset size, preprocessing techniques, and class balance should also be considered when interpreting these results.

Chapter 6

Limitations and Challenges

During the course of our research project, we encountered a number of challenges and limitations, the most significant of which was related to the dataset we used. We chose to work with the "Breast Histopathology Images" dataset from Paul Mooney, which is a large and complex dataset consisting of images of the most common subtype of breast cancer, Invasive Ductal Carcinoma (IDC).

Initially, we attempted to download the dataset in order to use it with Google Colab, the environment we used for our project. However, we soon discovered that the dataset was too large for our devices to handle. In order to overcome this challenge, we opted to use Kaggle, which we thought would provide an easier way to access and process the data. However, even with Kaggle, we faced a number of difficulties. For example, we struggled with downloading the data due to the numerous implementation options available, such as .walk or listdir(). Additionally, we had to become familiar with Kaggle itself, which was also challenging at first

Despite these challenges, we chose to work with the "Breast Histopathology Images" dataset because it still provided a valuable resource for exploring and analyzing histopathology images. While the images were not captured by us and may not have been the most accurate or up-to-date, they still represented a diverse range of cases and allowed us to test and refine our analytical methods. Furthermore, we acknowledged the limitations of using someone else's images and the ethical questions related to proper attribution and permission. Nonetheless, we believed that the benefits of using the dataset outweighed its limitations.

That said, it is important to note that every dataset has its limitations, and the careful selection and utilization of appropriate datasets are crucial for the development of reliable and clinically useful machine learning models. Therefore, as mentioned before, acknowledging the limitations and the unbalanced nature of our dataset is crucial in ensuring the accuracy and clinical utility of the deep learning models we develop for breast cancer detection. And for this reason, researchers must always carefully consider the limitations of any dataset they use and continue to explore and incorporate new data sources to improve the accuracy and robustness of machine learning models.

Pre-processing the data was another challenge that we encountered during our project. As we began exploring the data, we realized that we had to preprocess the images before feeding them into our models. This involved resizing the images, applying various filters to enhance the contrast, and splitting the dataset into training and testing sets. Pre-processing the data proved to be a challenge as we had to use

new techniques that we had not encountered during the course. However, we were able to overcome this challenge through extensive research and trial and error.

Another major challenge we faced was implementing four different machine learning models and keeping track of the progress and struggles with each one. We had to make sure that we were implementing the models correctly, and we had to keep data of all changes and optimizations made to the models. At first, this process was a struggle as we did not have a structured plan in place. However, as we continued to work together as a team, we were able to develop a more structured plan and process for implementing the models. Additionally, we had to find the strengths and weaknesses of each team member and leverage them accordingly to make sure everything worked smoothly.

As researchers, it is crucial to acknowledge our limitations, and we recognize that our study has its own set of limitations. While we presented a thorough comparison between different approaches to breast cancer detection using deep learning models, there may be other models or techniques that could be explored.

Chapter 7

Conclusion

In conclusion, breast cancer is a significant global health concern, and mammography testing remains the most effective tool for early detection. However, even highly skilled radiologists can miss breast cancers due to human error, distraction, and fatigue. As a team of three women, we chose to focus on breast cancer detection because we believe it is a crucial and compelling topic. Our motivation behind this project was to explore techniques for analyzing mammograms that can accurately detect and prevent late or unnecessary treatments.

Throughout our work, we have explored different deep learning models, including the ResNet50, VGG16, and InceptionV3, in combination with transfer learning. In order to establish a baseline for comparison, we also developed a basic convolutional neural network. Our experimentation has allowed us to gain a comprehensive understanding of how these models operate and which specific configurations lead to the most accurate detection and classification of breast cancer in mammograms. Through analysis and comparison, we were able to identify the models that showed the most promise and demonstrated the greatest potential to contribute to the field of breast cancer detection.

However, as researchers, we must acknowledge our limitations. Our study was limited by the size of the dataset we used and the computational resources available. Additionally, although we achieved high levels of accuracy, we recognize that there is still room for improvement.

Moving forward, our hope is that our work will contribute to the development of more accurate and efficient deep learning models for breast cancer detection. We believe that continued research and innovation in this field can lead to improved outcomes for patients and save lives. Ultimately, we are proud to have contributed to this important field and look forward to seeing the continued progress in breast cancer detection and treatment.

Chapter 8

Figures

Disclaimer: The code snippets presented, as well as the results plots, in the paper are all from our own code implementation, which can be found on our github: https://github.com/AliBenovaa/Breast-Cancer-Detection-Final-Project

1st Cover page: The logo of the University of Osnabrück on the cover page. Retrieved March 20,2023, from https://quantrix.com/quantrix-customers/university-of-osnabrueck/.

2.1 :A Basic CNN Model compared to the human visual cortex: Soffer, S., Klang, E., Shimon, O., & Tikvah, P. (2020, September). Figure 2. Schematic representation of the analogy between a CNN and a. . . ResearchGate. Retrieved March 19, 2023, from "https://www.researchgate.net/figure/Schematic-representation-of-the-analogy-between-a-CNN-and-a-biologic-visual-cortical_fig2_344329197".

2.2: Schematic representation of a VGG16:

2.3: Left: a building block for ResNet34. Right: a "bottleneck" building block for ResNet-50: He, K. (2015, December 10). Deep Residual Learning for Image Recognition. arXiv.org. Retrieved March 29, 2023 from "https://arxiv.org/abs/1512.03385v1".

2.4: A high-level diagram of the InceptionV3 model: Advanced Guide to Inception v3. (n.d.). Google Cloud. Retrieved March 20, 2023
from "https://cloud.google.com/tpu/docs/inception-v3-advanced"

2.5: One of the patches from the dataset labelled 10253_idx5_x1001_y1251_class0: Mooney,P.(2018). Retrieved March 20,2023, from "https://www.kaggle.com/datasets/paultimothymooney/breast-histopathology-images"

3.1:Daataaugmentation for the Data preparation

3.1: code snippet for resizing the image

3.1: Matplotlib output exhibiting example images of IDC(+) and IDC(-) from our code

3.2: A part of the implemented CNN model to visualise the implementation of initialising the model sequential and adding a Conv2D, MaxPool and Dropout Layer.

3.2: A part of the implemented CNN model to visualise the implementation of adding the flatten and dense layers to the model.

3.2: Graph output of CNN model without data augmentation(left) and with data augmentation (right)- left image is displayed in our github

3.3: Snippet of the Inception V3 model with our own classifier and input image size from our code

3.4: Comparing the results from both optimizers with the RMSprop optimizer on the left- left graph is in the github

3.5:Graph output of ResNet50 model with learning rate 0.001 (right) and 0.0001 (left) - left graph is on our github

3.3.2: Code snippet of reducing the sample size for kaggle storage

3.3.3: A part of the implemented Inception V3 model to visualise the implementation of initialising the model and adding early stopping technique.

3.6: The result graph after running the InceptionV3 model for the first time. Noticing that it is stuck around 85 %.

3.7 Resize the images for optimal use for InceptionV3 model.

4.1: Graph representing the results of the CNN

4.1:A Basic Model of a Confusion Matrix based on the information: Handoko, N. T. (2022, July 27). Confusion Matrix Explained - Towards Data Science. Medium. Retrieved March 19, 2023, from https://towardsdatascience.com/confusion-matrix-explained-5d42122a04d6.

4.2: Confusion Matrix of the CNN Model displayed in our github

2.4: A graph representing the model architecture based on the google cloud section "Advanced Guide to Inception v3 " : https://cloud.google.com/tpu/docs/inception-v3-advanced

4.3: Receiver Operating Characteristic (ROC) Curve for CNN Model displayed in github

4.4: Histogram of prediction error for CNN model displayed in our github

4.5:Neural network training results: 90% accuracy achieved in 10 epochs with an average speed of 23 seconds per epoch displayed in the github

Bibliography

[1] D. Sujitha Juliet A. Sai Bharadwaj Reddy. *Transfer Learning with ResNet-50 for Malaria Cell-Image Classification*. URL: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=8697909.

[2] Adam Jaamour. *Breast Cancer Detection in Mammograms using Deep Learning Techniques*. URL: https://info.cs.st-andrews.ac.uk/student-handbook/files/project-%20library/cs5098/agj6-Final_report.pdf.

[3] Al-Haija, Q. A. Adebanjo, A. *Breast Cancer Diagnosis in Histopathological Images Using ResNet-50 Convolutional Neural Network*. URL: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=9216455.

[4] Mera-Jimenez,L. Zequera-Diaz,M. Anaya-Isaza,A. *An overview of deep learning in medical imaging*. URL: https://www.sciencedirect.com/science/article/pii/S2352914821002033.

[5] Boesch,G. *Deep Residual Networks (ResNet, ResNet50) 2023 Guide*. URL: https://viso.ai/deep-learning/resnet-residual-neural-network/.

[6] Das, S. *CNN Architectures: LeNet, AlexNet, VGG, GoogLeNet, ResNet and more*. URL: https://medium.com/analytics-vidhya/cnns-architectures-lenet-alexnet-vgg-googlenet-resnet-and-more-666091488df5.

[7] Fawcett,T. *An introduction to ROC analysis. Pattern Recognition Letters*. URL: https://www.sciencedirect.com/science/article/abs/pii/S016786550500303X.

[8] Kalluri,H.K. Krishna,S.T. *Deep Learning and Transfer Learning Approaches for Image Classification. International Journal of Advanced Science and Technology*. URL: https://www.researchgate.net/profile/Hemantha-Kumar-Kalluri/publication/333666150%5C_Deep%5C_Learning%5C_and%5C_Transfer%5C_Learning%5C_Approaches%5C_for%5C_Image%5C_Classification/links/5cfbeeb9a6fdccd1308d6aae/Deep-Learning-and-Transfer-Learning-Approaches-for-Image-Classification.pdf.

[9] Lenin G. Falconi, Maria P´erez, Wilbert G. Aguilar, Aura Conci. *Transfer Learning and Fine Tuning in Breast Mammogram Abnormalities Classification on CBIS-DDSM Database*. URL: https://www.astesj.com/v05/i02/p20/#1477470858207-d194995f-690d.

[10] Mueller, V.Scharl,A Lüftner,D. *Wie häufig ist Brustkrebs?* 2022. URL: https://www.krebsgesellschaft.de/onko-internetportal/basis-informationen-krebs/krebsarten/brustkrebs-definition-und-haeufigkeit.html.

[11] Mallick, S. *Transfer Learning for Medical Images — LearnOpenCV . LearnOpenCV – Learn OpenCV, PyTorch, Keras, Tensorflow With Examples and Tutorials.* URL: https://learnopencv.com/transfer-learning-for-medical-images/#Should-you-use-ImageNet-architectures-for-solving-medical-problems?.

[12] Mikulskibartosz. *How to visualise prediction errors.* URL: hhttps://www.mikulskibartosz.name/how-to-visualise-prediction-errors/.

[13] Leonie Monigatti. *Intermediate Deep Learning with Transfer Learning.* URL: https://towardsdatascience.com/intermediate-deep-learning-with-transfer-learning-f1aba5a814f.

[14] P. Mooney. *Breast Histopathology Images.* URL: https://www.kaggle.com/datasets/paultimothymooney/breast-histopathology-images.

[15] n.a. *Advanced Guide to Inception v3.* URL: https://cloud.google.com/tpu/docs/inception-v3-advanced.

[16] n.a. *InceptionV3.* URL: https://keras.io/api/applications/inceptionv3/.

[17] n.a. *ResNet-50.* URL: https://www.kaggle.com/datasets/keras/resnet50.

[18] n.a. *The Global Cancer Observatory Bulgaria.* 2021. URL: https://gco.iarc.fr/today/data/factsheets/populations/100-bulgaria-fact-sheets.pdf.

[19] n.a. *Types of Breast Cancer.* URL: https://www.macmillan.org.uk/cancer-information-and-support/breast-cancer/types-of-breast-cancer.

[20] n.a. *UK cancer statistics.* URL: https://www.wcrf-uk.org/preventing-cancer/uk-cancer-statistics/#:~:text=Most%5C%20common%5C%20cancers%5C%20in%5C%20the%5C%20UK%5C%20%5C%20,%5C%20%5C%2044%5C%2C706%5C%20%5C%2011%5C%20more%5C%20rows%5C%20.

[21] Poonam Verma. *Comparison of different optimizers implemented on the deep learning architectures for COVID-19 classification.* URL: https://www.sciencedirect.com/science/article/pii/S2214785321013316.

[22] Prasad Pai. *Data Augmentation Techniques in CNN using Tensorflow.* URL: https://medium.com/ymedialabs-innovation/data-augmentation-techniques-in-cnn-using-tensorflow-371ae43d5be9.

[23] Rohini G. *Everything you need to know about VGG16.* URL: https://medium.com/@mygreatlearning/everything-you-need-to-know-about-vgg16-7315defb5918.

[24] A.Ponti J. Pattnick S. Hofveind. *False-positive results in mammographic screening for breast cancer in Europe: a literature review and survey of service screening programmes.* URL: https://pubmed.ncbi.nlm.nih.gov/22972811/.

[25] S. Saha. *A Comphrehensive Guide to Convolutional Neural Networks - the ELI5 way.* URL: https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53.

[26] Shadid,T. *ResNet: A Simple Understanding of the Residual Networks.* URL: https://medium.com/swlh/resnet-a-simple-understanding-of-the-residual-networks-bfd8a1b4a447.

[27] Karen Simonyan and Andrew Zisserman. “Very Deep Convolutional Networks for Large-Scale Image Recognition”. In: *International Conference on Learning Representations*. 2015.