Breast Cancer Classification

Syed Ibtehaj Raza, Rizvi   Faiza Tahsin   Zarin Tasnim

Abstract

Breast cancer is a deadly disease but we can reduce the death risk and increase the possibility of survival chances and a successful treatment by detecting it in it's early stages. In this project, with the help of machine learning (ML) algorithms we are trying to differentiate between Non-invasive Ductal Carcinoma (non-IDC) and Invasive Ductal Carcinoma (IDC) tumors based on the abnormal breast tissues. IDC is one of the most common form of breast cancer and are trying to classify it by using feature analysis from histopathological image data-set, using different variations of deep convolutional neural networks (CNN). For this purpose we are using three different CNN models, namely, RestNet34, Inception-V3 and VGG16 to classify IDC and non-IDC.

1. Introduction

Sometimes, the expert radiologists can fail to detect the features of breast cancer which can lead to a false-positive or more dangerously false negative diagnosis. In this project our aim is to successfully identify cancerous tumor type using the selected data-set to classify between Noninvasive Ductal Carcinoma (non-IDC) and Invasive Ductal Carcinoma (IDC) tumors with the help machine learning models. For this purpose we have implemented three different types of ML algorithms. After successful implementation and training we are presenting our results here.

2. Dataset

One of the most important part of any ML project is the data-set. For that, we are using a histopathological image data-set which we downloaded from kaggal[1] but it's originally provided by Gleaso Institute for Neuroscience. The original dataset consisted of 162 whole mount slide images of Breast Cancer (BCa) specimens. From that, 277,524 patches were extracted (198,738 IDC negative and 78,786 IDC positive). An example of the data can be seen in Figure 1. Because of the size of the available data, utilizing the full data-set for training purposes is out of the scope of this project. So, in order to find the best possible number of samples which can give us good results we ran some pre-trained models on multiple data sizes with multiple configurations of hyper-parameters. After analyzing the results we chose to go with the data-set of 2000 IDC and 2000 non-IDC images to train our self-implemented model with the split of training, testing and validation sets in the 80:10:10 ratio. As this was raw data, the images needed to be normalised and resized to 224 by 224 with 3 number of channels.

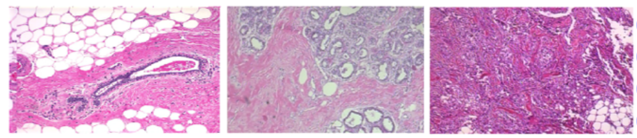

Figure 1. Example of patches of images.

3. Methodology

Our aim in this project, other than classification, was to maximize our understanding of end-to-end ML life cycle from data collection to implementing and training the model. In the light of that, each member of the group have trained one model of our choice which are ResNet34, Inception-V3 and VGG16.

3.1. ResNet

One of the most common notion associated with CNN is "the deeper the better". Even though this makes complete sense because the model should be more capable. However, we have noticed that after some depth the performance of the model degrades. Turns out as we add more layers using certain activation functions to our neural networks, the gradients of the loss function approaches zero, making the network hard to train. This phenomenon is also known as **Vanishing Gradient** problem.

Residual Networks (ResNet) solve this problem in a novel manner, Neural networks are good function approximators, they should be able to easily solve the identify function, where the output of a function becomes the input itself (1) :

$$f(x) = x \tag{1}$$

Using the logic, by bypassing the input of the first layer to the output of the last layer of the model the network should be able to predict whatever function it was learning before with the input added to it (2). This is the intuition behind the ResNet.

$$f(x) + x = h(x) \tag{2}$$

The basic ResNet architecture consist of two things, blocks 3 and layers.

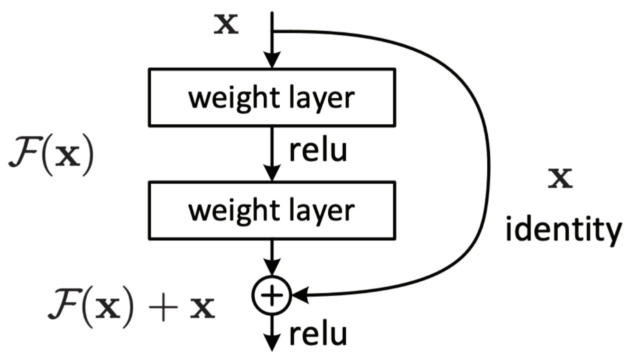

Figure 2. Residual learning: a building block

A block is the first step of the ResNet before entering the common layer behavior, which consists on a convolution, batch normalization and max pooling operation. Next are the ResNet Layers which are made by repeating these blocks. Each layer is made up of several blocks and an operation here refers to a convolution a batch normalization and a ReLU activation to an input, except the last operation of a block, that does not have the ReLU. The resNet architecture can be seen here 3

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

Figure 3. ResNet Architecture

3.1.1 ResNet-34

The author [2] proposed multiple architecture 3 with almost same building blocks. For the purpose of this project we am using the 34 layer architecture due to less number of layers then the higher once. One of the biggest difference between ResNet34 and ResNet with more layers(50, 101, etc) is the depth of the layer. In ResNet34, two layer deep block is used while 3 layer deep block is used for the higher once (50, 101, etc).

3.2. Inception

Inception-v3 belongs to the Inception family which is also a convolutional neural network architecture which enables many modifications, including the use of Label Smoothing, Factorized 7 x 7 convolutions, and the use of an auxiliary classifier to relay label information through the network. According to the author[3], Inception networks are fully convolutional where each weight corresponds to one multiplication per activation. Therefore, any reduction in computational cost can reduce the number of parameters which can result in faster training. In our implemented Inception-V3 model we have made some modifications in the network and added a few layers to make the model efficient so that it can generalize our data-set.

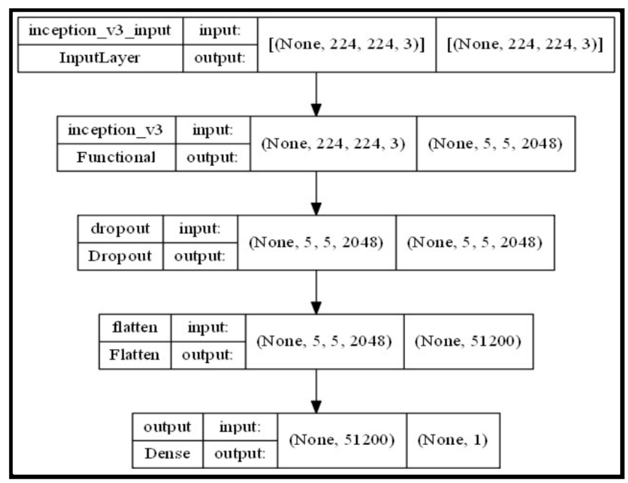

Figure 4. Architecture of our Inception-V3 model

3.3. VGG16

VGG16 is a Convolutional Neural Network that is 16 layers deep. Instead of having a large number of hyper-parameter they focused on having convolution layers of 3x3 filter with a stride 1 and always used same padding and maxpool layer of 2x2 filter of stride 2. The idea behind using 3 x 3 filters uniformly is something that makes the VGG stand out. Two consecutive 3 x 3 filters provide for an effective receptive field of 5 x 5. Similarly, three 3 x 3 filters make up for a receptive field of 7 x 7. This way, a combination of multiple 3 x 3 filters can stand in for a receptive area of a larger size. In addition to the three convolution layers, there are also three non-linear activation layers instead of a single one you would have in 7 x 7. This makes the decision functions more discriminative. It would impart the ability to the network to converge faster. It follows this arrangement of convolution and max pool layers consistently throughout the whole architecture. In the end it has 2 FC (fully connected layers) followed by a softmax for output. This network is a pretty large network and it has about 138 million (approx) parameters.

4. Experimental Results

Due to the nature of the project and our approach, this section will be divided into four parts. One part for each model discussing the results and the last part for the result comparison.

4.1. ResNet-34

We started the ResNet34 implementation by analyzing the data-set. As mentioned before we have a fairly big data-set which is divided into multiple folder according to the patient names. Before starting the model implementation we needed to finalize the data-set size and the appropriate hyper-parameters. For this purpose we implemented a quick pre-trained model and start optimizing our parameters. Most of the data-reprocessing and hyper-parameter selection is done with the help of a pre-trained model provided by keras. We have used a hybrid set-up of google CoLabs and our local environment for the training. As stated earlier for the CNN to work we needed to needed to normalize and resize the images. Furthermore, we have also implemented horizontal and vertical flipping to make the model more generalize. The results of the pre-trained models can be seen in the appendix.

After getting those parameters using pre-trained model we implemented a model from scratch. The implementation consist on two parts data-processing and preparation, and the actual implementation of the model. ResNet implementation is divided into two parts. The block and the layers. The block, as stated above, consist on convolution, batch normalization and max pooling operation. For ResNet-34 the layer structure is [3, 4, 6, 3] which can be seen in the architecture diagram(3) as well. I have coded the ResNet class in such a way that it can be adapted for other layers as well because that part is dynamic. As a loss function we are using binary_crossentropy function which computes the cross-entropy loss between true labels and the predicted labels. The results for the self-implemented models can be seen here:

```
Epoch: 048/050 training accuracy: 92.94% | Validation accuracy: 94.53%
Time elapsed: 1442.36 min
Epoch: 049/050 | Batch 000/026 | Cost: 0.2171
correct_pred:  tensor(2958) , num_examples:  3201
correct_pred:  tensor(192) , num_examples:  201
Epoch: 049/050 training accuracy: 92.41% | Validation accuracy: 95.52%
Time elapsed: 1472.49 min
Epoch: 050/050 | Batch 000/026 | Cost: 0.1688
```

Figure 5. ResNet Train Data Accuracy

As you can see, we have been able to achieve a good training accuracy of 92.41 percent and testing accuracy of 92.17 percent. After training, we are also saving the model for future use and project evaluation.

```
correct_pred:  tensor(553) , num_examples:  600
Test accuracy: 92.17%
```

Figure 6. ResNet Test Data Accuracy

	precision	recall	f1-score	support
0	0.92	0.99	0.95	72
1	0.91	0.62	0.74	16
accuracy			0.92	88
macro avg	0.92	0.81	0.85	88
weighted avg	0.92	0.92	0.91	88

Figure 7. ResNet Classification Report

4.2. Inception

For Inception-V3, We have freeze some layers and added new layers to the network and trained it. While implementing inception-V3 we have made the layer non-trainable except for the last 3 layers to avoid destroying any of the information they contain during future training rounds. Then we have added some trainable layers on top of frozen layers. We have resized the image as 224x224x3 and fed that image into the network. Then we have added a dropout of 0.5 to prevent the model from over-fitting. After that we have used flatten for converting the data into a 1-dimensional array for inputting it to the next layer. Finally we have used a dense layer where we have used sigmoid activation function 8 for prediction. We have used Model Checkpoint and Early stop in order to monitor the validation accuracy and save the model depending on the improvement. If the validation accuracy degrades in the current epoch and it does not improve in the following 5 epochs (Patience = 5) then the training will stop and the model will be saved. We have used early stop in order to avoid over-fitting and under-fitting. Adam is used as the optimizer with a learning rate of 1e-3. We have used binary cross entropy as our loss function and sigmoid as activation function because we have two classes. The result of the model can be seen here:

$$f(x) = \frac{1}{1 + e^{-x}}$$

Figure 8. Sigmoid Activation Function

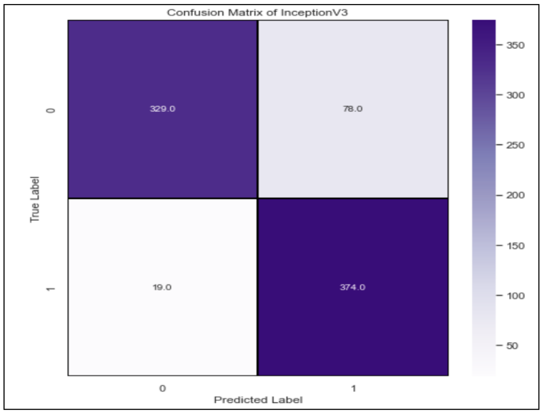

Figure 9. Inception Confusion Matrix Architecture

	precision	recall	f1-score	support
0	0.95	0.81	0.87	407
1	0.83	0.95	0.89	393
accuracy			0.88	800
macro avg	0.89	0.88	0.88	800
weighted avg	0.89	0.88	0.88	800

Figure 10. Inception Report

Figure 11. Plot diagram of train loss and validation loss(left) and train accuracy and validation accuracy

4.3. VGG16

In implementation, first we loaded the VGG16 model with ImageNet trained weights and pass our input shape as (224, 224, 3) which is our base model. Then, we made all loaded layers as non-trainable. This is important as we wanted to work with pre-trained weights. After that, we made our new custom sequential model. Where, we add all layers from our base model except the output layer. Generally, The shape of output layer is 1000 and the activation function is softmax. According to our dataset the last dense layer should have shape of 1 and Sigmoid as an activation function. In the compilation of the model, as an optimiser we use Adam optimiser to reach to the global minima while training our model. If we stuck in local minima while training then the Adam optimiser will help us to get out of local minima and reach global minima. We will also specify the learning rate of the optimiser, here in this case it is set at 0.001. If our training is bouncing a lot on epochs then we need to decrease the learning rate so that we can reach global minima. As a loss function we use binary_crossentropy function that computes the cross-entropy loss between true labels and predicted labels. And metrics will be accuracy.

ModelCheckpoint and EarlyStopping are the callbacks function while training the model.

ModelCheckpoint - It helps us to save the model by monitoring a specific parameter of the model. In this case I am monitoring validation accuracy by passing val_accuracy to ModelCheckpoint. The model will only be saved to disk if the validation accuracy of the model in current epoch is greater than what it was in the last epoch.

Early-Stopping - It helps us to stop the training of the model early if there is no increase in the parameter which I have set to monitor in EarlyStopping. In this case I am monitoring validation accuracy by passing val_accuracy to EarlyStopping. I have here set patience to 5 which means that the model will stop to train if it doesn't see any rise in validation accuracy in 5 epochs. In the training, we feed

```
Epoch 8: val_accuracy did not improve from 0.79062
45/45 - 287s - loss: 0.4535 - accuracy: 0.8118 - val_loss: 0.5079
- val_accuracy: 0.7844 - 287s/epoch - 6s/step
Epoch 8: early stopping
```

Figure 12. Training Outputs

the model with training dataset and validation dataset where epochs were 100 with the above callbacks function. After 8 epochs our training stops because the last 5 epochs weren't improving our validation accuracy.

```
Epoch 8: val_accuracy did not improve from 0.79062
45/45 - 287s - loss: 0.4535 - accuracy: 0.8118 - val_loss: 0.5079
- val_accuracy: 0.7844 - 287s/epoch - 6s/step
Epoch 8: early stopping
```

Figure 13. Training Outputs

```
13/13 - 70s - loss: 0.4926 - accuracy: 0.8062 - 70s/epoch - 5s/step
Accuracy: 80.62%
Loss: 49.26%
```

Figure 14. Model evaluation on test data

4.4. Comparison

As we it can be seen in the classification report from each implemented model the accuracies in all the implemented models are competitive. We have achieve the highest accuracy with the ResNet model which is also supported by

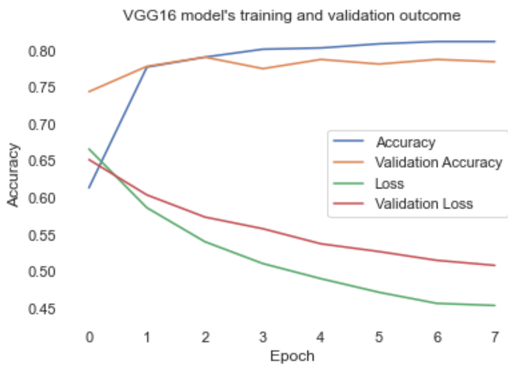

Figure 15. Training and validation outcomes

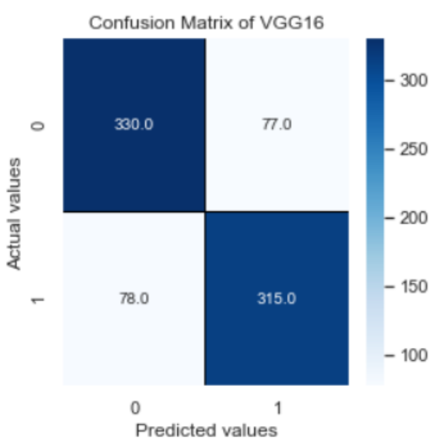

Figure 16. Confusion matrix

	precision	recall	f1-score	support
0	0.81	0.81	0.81	407
1	0.80	0.80	0.80	393
accuracy			0.81	800
macro avg	0.81	0.81	0.81	800
weighted avg	0.81	0.81	0.81	800

Figure 17. Classification report

multiple papers. Other models were also able to achieve more then 80% accuracy level which shows the success of this project.

5. Conclusions

To sum it up, each of the chosen model was able to classify the images with the good accuracies. However, results can be improve even more by utilizing the full data-set and opting for more complex models with more in-depth layers. More results and work break-down can be found in the appendix section.

	precision	recall	f1-score	support
0	0.92	0.99	0.95	72
1	0.91	0.62	0.74	16
accuracy			0.92	88
macro avg	0.92	0.81	0.85	88
weighted avg	0.92	0.92	0.91	88

Figure 18. ResNet Classification Report

	precision	recall	f1-score	support
0	0.95	0.81	0.87	407
1	0.83	0.95	0.89	393
accuracy			0.88	800
macro avg	0.89	0.88	0.88	800
weighted avg	0.89	0.88	0.88	800

Figure 19. Inception Report

	precision	recall	f1-score	support
0	0.81	0.81	0.81	407
1	0.80	0.80	0.80	393
accuracy			0.81	800
macro avg	0.81	0.81	0.81	800
weighted avg	0.81	0.81	0.81	800

Figure 20. Classification report

References

[1] Gleason Institute for Neuroscience. Breast histopathology images. 1

[2] Shaoqing Ren Kaiming He, Xiangyu Zhang and Jian Sun. Deep residual learning for image recognition. *arxiv*, 2015. 2

[3] Kurama and Vikar. A review of popular deep learning architectures: Resnet, inceptionv3, and squeezenet. 2019. 2

Appendix: Extra Content

In this section we are including the work break-down and other results which we got while training our models. The breakdown can be seen in the table 1

.1. ResNet-34 Pre-Trained Model

```
Epoch 1/10
1318/1318 [==============================] - 938s 708ms/step - loss: 0.5993 - acc: 0.8369 - val_loss: 0.3178 - val_acc: 0.8561
Epoch 2/10
1318/1318 [==============================] - 760s 577ms/step - loss: 0.2941 - acc: 0.8809 - val_loss: 1.0771 - val_acc: 0.8052
Epoch 3/10
1318/1318 [==============================] - 770s 584ms/step - loss: 0.2712 - acc: 0.8839 - val_loss: 0.7589 - val_acc: 0.8295
Epoch 4/10
1318/1318 [==============================] - 770s 585ms/step - loss: 0.2414 - acc: 0.8998 - val_loss: 0.3907 - val_acc: 0.8565
Epoch 5/10
1318/1318 [==============================] - 752s 571ms/step - loss: 0.2155 - acc: 0.9097 - val_loss: 0.5551 - val_acc: 0.8481
Epoch 6/10
1318/1318 [==============================] - 765s 581ms/step - loss: 0.1871 - acc: 0.9196 - val_loss: 0.5520 - val_acc: 0.8523
Epoch 7/10
1318/1318 [==============================] - 797s 605ms/step - loss: 0.1863 - acc: 0.9234 - val_loss: 0.7146 - val_acc: 0.8398
Epoch 8/10
1318/1318 [==============================] - 771s 585ms/step - loss: 0.1465 - acc: 0.9401 - val_loss: 0.7496 - val_acc: 0.8409
Epoch 9/10
1318/1318 [==============================] - 778s 590ms/step - loss: 0.1413 - acc: 0.9416 - val_loss: 0.6143 - val_acc: 0.8565
Epoch 10/10
1318/1318 [==============================] - 796s 605ms/step - loss: 0.1282 - acc: 0.9484 - val_loss: 0.7525 - val_acc: 0.8497
```

Figure 21. ResNet Pre-Trained Model Training Output

```
83/83 [==============================] - 480s 6s/step - loss: 0.7364 - acc: 0.8462
Accuracy: 84.62%
```

Figure 22. ResNet Pre-Trained Model Accuracy on Testing Data

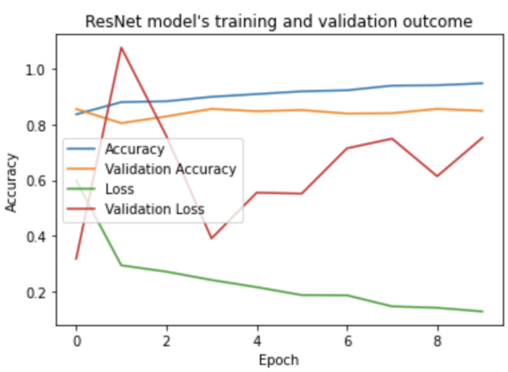

Figure 23. ResNet Pre-Trained Model Graph

	Responsible Team Member	Work Done by
ResNet-34	Syed Ibtehaj Raza, Rizvi	Self
VGG-16	Faiza Tahsin	Self
Inception-V3	Zarin Tasnim	Self
Presentation	All of us	Presented and created by Ibtehaj with the help of Faiza and Zarin
Report	All of us	Created by Ibtehaj with the help of Faiza and Zarin.

Table 1. Work breakdown for the project.