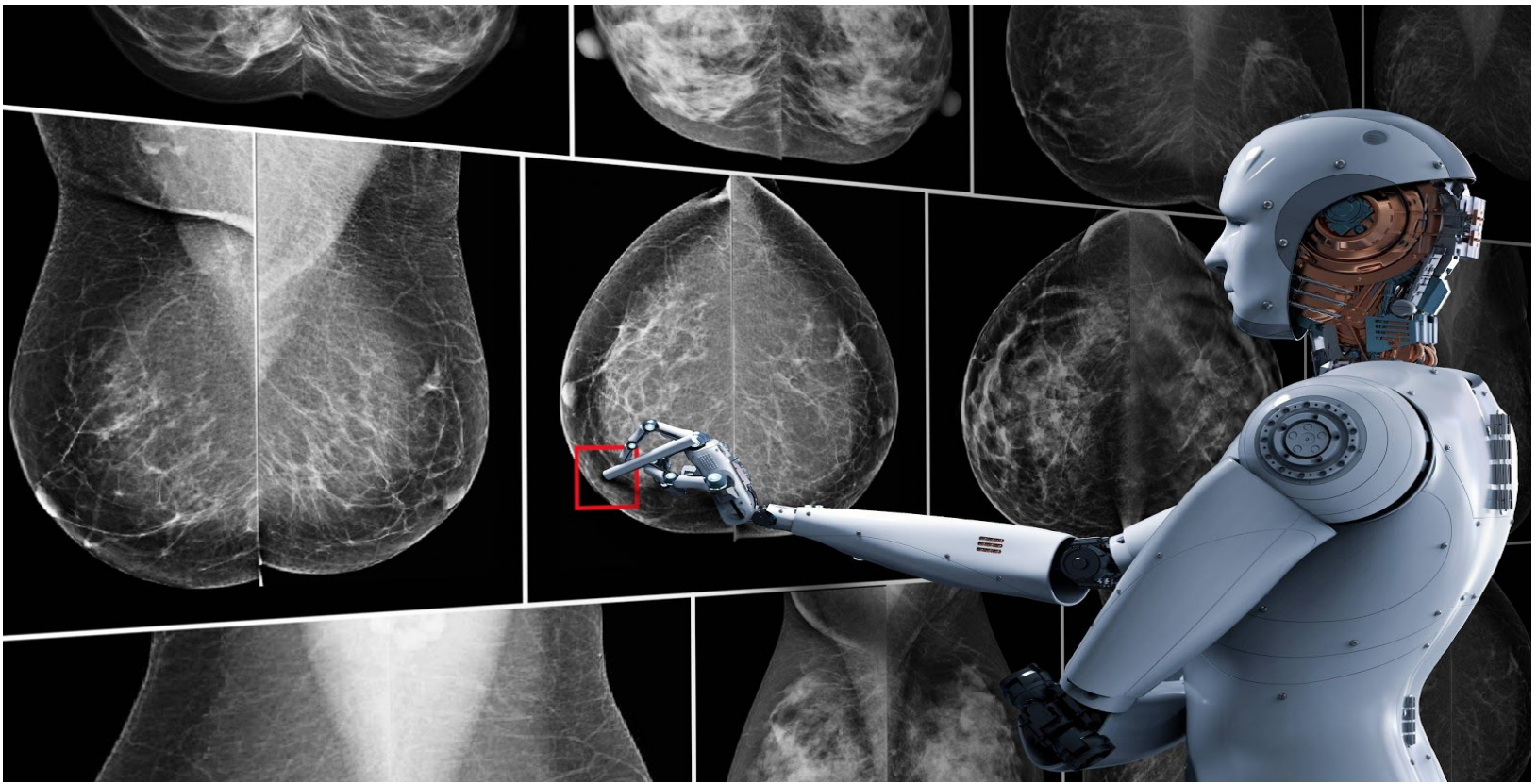
Breast Cancer (IDC) Detection

Machine Learning Capstone Project Report

By- Animesh Seemendra
Machine Learning Advance Nanodegree
Date- 29/12/2018



Definition

Project Overview

Breast Cancer is the most common type of cancer in women worldwide accounting for 20% of all cases. In 2012 it resulted in 1.68 million new cases and 522,000 deaths. One of the major problems is that women often neglect the symptoms, which could cause more adverse effects on them thus lowering the survival chances. In developed countries, the survival rate is although high, but it is an area of concern in the developing countries where the 5-year survival rates are poor. In India, there are about one million cases every year and the five-year survival of stage IV breast cancer is about 10%. Therefore it is very important to detect the signs as early as possible.

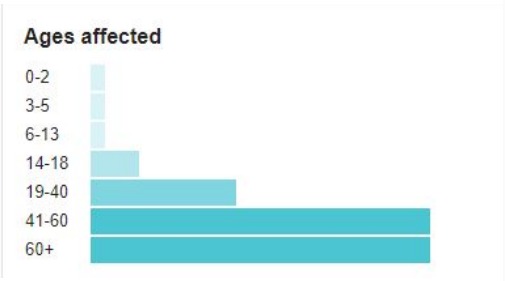


Fig 1

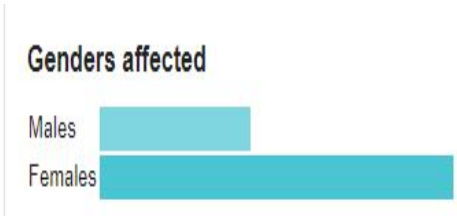


Fig 2

The data is on Breast cancer in India taken from various hospitals (Apollo and others)

Invasive ductal carcinoma (IDC) is the most common form of breast cancer. About 80% of all breast cancers are invasive ductal carcinomas. Doctors often do the biopsy or a scan if they detect signs of IDC. The cost of testing for breast cancer sets one back with $5000, which is a very big amount for poor families and also manual identification of presence and extent of breast cancer by a pathologist is critical. Therefore automation of detection of breast cancer using Histopathology images could reduce cost and time as well as improve the accuracy of the test. This would be the approach that we would be looking forward to accomplishing in this project.

This is the active research field, many scientists are working on this to automate the system of breast cancer detection. Many papers have been published on this topic like https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5453426/ .

Therefore applying deep learning techniques such as CNN architecture, transfer learning and image argumentations could help us to achieve our result.

The problem statement involves using images that are obtained during pathology tests to detect whether the patient is IDC positive or negative. Histopathology images are the images of tissues that are obtained during pathology tests, therefore, these images will act as inputs for the problem.

The dataset that contains histopathology images for breast cancer is present on kaggle at : https://www.kaggle.com/paultimothymooney/breast-histopathology-images . The original dataset consisted of 162 whole mount slide images of Breast Cancer (BCa) specimens scanned at 40x. However, the data that I have selected contains images that are cropped from the original dataset i.e it contains patches of regions where the IDC occurs, making it more specific to our problem.

Problem Statement

The idea is to use pathology test images and classify them as IDC(+) and IDC(-). Accurately identifying and categorizing breast cancer subtypes is an important clinical task, and automated methods can be used to save time and reduce error. The pathological tests include images of the tissues, the task is to train a computer to use these images and respond on whether the person is IDC(+) or IDC(-). Since it is a medical field problem it is important that sensitivity of the output should be high.

Our data involves images with the classes written on data file name, therefore, we would need to extract the class name from it and create a column to store them. We also need to split the dataset into the training set, validation set and testing set. Testing set for checking how good the model works on completely unseen data and validation set to check and avoid underfit or overfit, the will also help to select the best model. One hot encoding will be done in classes column so that it could work better with our model. Image processing step is also required to reduce the pixel range from 0-250 to 0-1. After it CNN model is to be used to predict the class, CNN creates an effective architecture the 2D structure of the image, therefore, it would be the best to use, considering that we are working with the images.

Evaluation Metrics

The performance of the model will be evaluated using ROC curve and confusion matrix. A receiver operating characteristic curve, i.e., ROC curve, is a graphical plot that illustrates the diagnostic ability of a binary classifier system as its discrimination threshold is varied. The ROC curve is created by plotting the true positive rate (TPR) against the false positive rate (FPR) at various threshold settings. The true-positive rate is also known as sensitivity, recall or probability of detection in machine learning. The false-positive rate is also known as the fall-out or probability of false alarm and can be calculated as (1 − specificity). It uses the concept of true positive, true negative, false positive and false negative.

$\text{Sensitivity} = \frac{\text{True Positive}}{\text{True Positive + False Negative}}$.	$\text{Recall} = \frac{\text{True Positive}}{\text{True Positive + False Negative}}$.
$\text{Specificity} = \frac{\text{True Negative}}{\text{True Negative + False Positive}}$.	$\text{Precision} = \frac{\text{True Positive}}{\text{True Positive + False Positive}}$.

The perfect classification has the area under the ROC curve equal to 1. Therefore closer the area of our ROC curve to 1 better would be our model. The third is a confusion matrix, it is a two by two table that contains four outcomes produced by a binary classifier. Various measures, such as error-rate, accuracy, specificity, sensitivity, and precision, are derived from the confusion matrix. Sensitivity can be calculated from the confusion matrix, which is important to know when we work in the medical domain i.e how many of the patients were told about having breast cancer our of how many were actually having it.

The ROC curve and confusion matrix would be a good evaluation matrix because they both are used for binary classification and our data is also based on binary classification. These metrics could help in evaluating the model through sensitivity, specificity, recall and precision which all are important and are always considered while working in this domain with it would provide us with the visualization of the correctness of the model.

Analysis

Data Exploration and Visualization

The dataset that contains histopathology images for breast cancer with 198,738 of negative IDC and 78,786 of positive IDC, therefore, it is a good dataset with enough data for our task. All images present in the data are of different dimensions, most of them are (50 X 50) while others have one or both dimension less than 50 which means that images are small. The dataset contain folders inside which various tissue images are present which are .png file.

Each patch's file name is of the format: u_xX_yY_classC.png — > example 10253_idx5_x1351_y1101_class0.png . Where u is the patient ID (10253_idx5), X is the x-coordinate of where this patch was cropped from, Y is the y-coordinate of where this patch was cropped from, and C indicates the class where 0 is non-IDC and 1 is IDC.

Image contains more brighter region than the darker region, this can be seen in the plot of color ranges.

The labels when extracted would be in categorical form. The analysis of figure 3 can be made that all the colours present mostly lie in the brighter region.

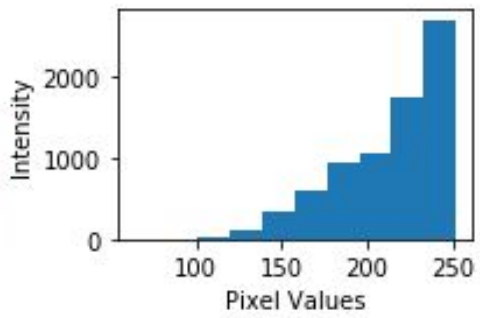


Figure 3 :Histogram Plot of an image pixels from the data

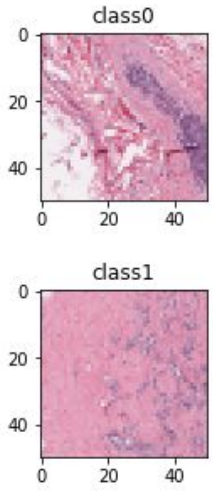


Figure 4 : `The images are from the data: class0 represents IDC(-) and class1 represents IDC(+)`

Class Distribution:

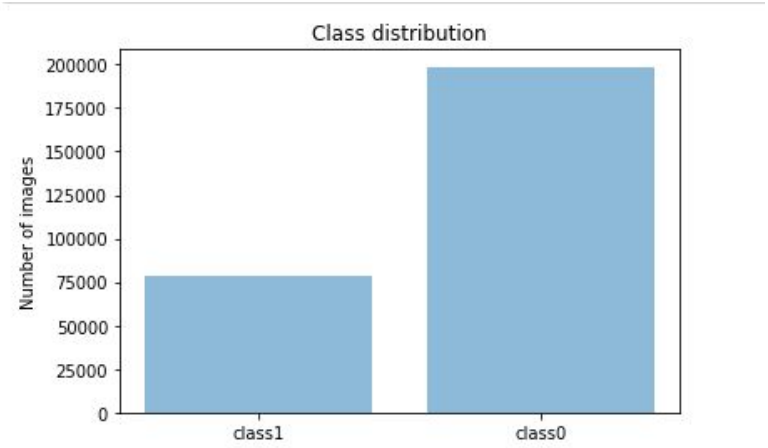


Figure 5: :Since we are working on medical domain class0 i.e IDC(-) images are more common than class1 (IDC(+)) images.

In figure 5 you can see that we have 71.6% of IDC(-) images and 28.3 % of IDC(+) images.

Conclusion: We would have to take care of such imbalances in our data and we would also not be removing color channels from images as it could be important for the analysis considering we are working with tissue images.

Algorithm and Techniques

The classifier is **Convolutional Neural Networks** , which is a state of the art algorithm for most image processing tasks, including classification. For better training lager data is required, luckily our data is large enough even after undersampling and data reduction.

Convolutional Neural Networks

CNNs are the most popular type of networks for image processing tasks. Unlike Dense or fully connected layers it retains the 2D structure of the image while learning. CNN was inspired by the visual cortex. Every time we see something, a series of layers of neurons gets activated, and each layer will detect a set of features such as lines, edges. The high level of layers will detect more complex features in order to recognize what we saw.

The input data is an image containing height, width and depth. It is a matrix of pixel values.The depth represents the channels (R, G, B). Depth is 1 in case of a greyscale image. The CNN works in the following ways :

1. It takes an image, which computer sees in the form of a matrix of pixel values.

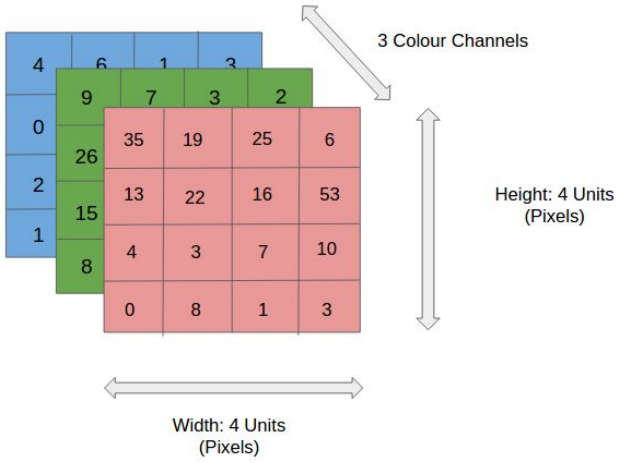


Figure 6: Input Image in form of a matix

2. The convolutions have filters, which are tiny squares that slide over the whole image, capturing the most striking parts.

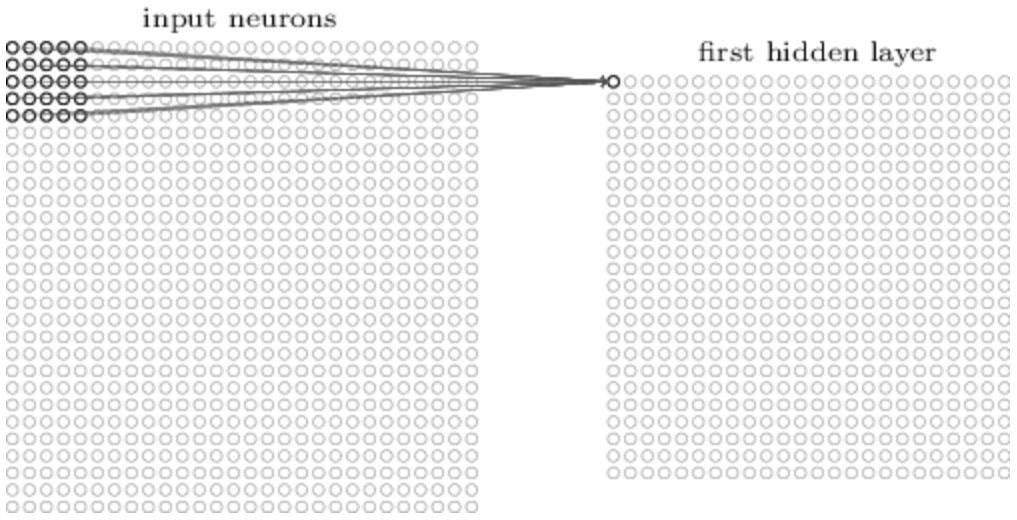


Figure 7: Filters sliding over an image

3. In figure 7: 32x32x3 image and a filter that covers a 5x5 image area and two-step motion (called stride), the filter will pass through the entire image, in each one of the channels, creating at the end a 28x28x1 feature map or activation map.

4. The convolution output depth is equal to the number of applied filters. The deeper the convolutional layers, the more detailed are the features identified by the activation map.

5. A layer is made up of a number of filters. And a model can contain a number of layers, depending upon how deeply you want to extract information from the images.

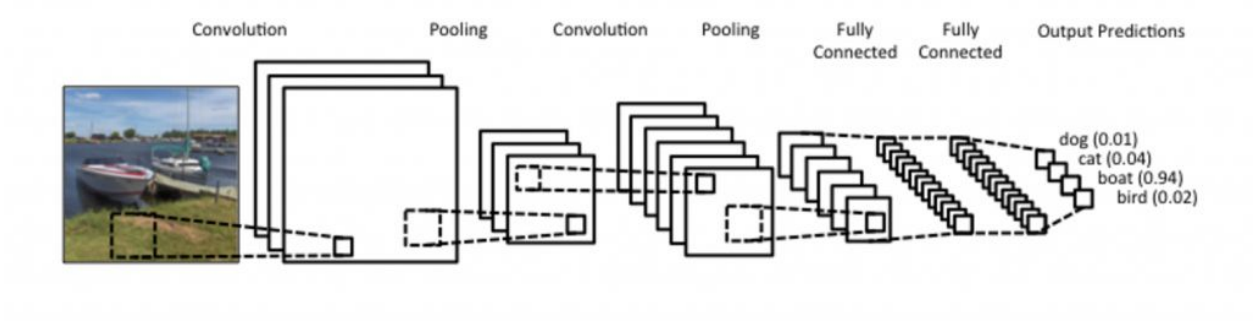


Figure 8: Understanding CNN

6. The filter is formed by randomly initialized weights.

7. There two processes that are involved while training a CNN, foward pass and backpropagation.

8. During a forward pass, a matrix passes through these layers and finally, the prediction is made which is compared to the actual output and according to the error of prediction from the output, backpropagation takes place which updates all the weights of the filters so that it gives the correct output .

Now we have a general understanding of how the CNN works, now let dive into some technical terms.

- **Forward Propagation :**

$$\begin{bmatrix} O_{11} & O_{12} \\ O_{21} & O_{22} \end{bmatrix} = \text{Convolution}\left(\begin{bmatrix} X_{11} & X_{12} & X_{13} \\ X_{21} & X_{22} & X_{23} \\ X_{31} & X_{32} & X_{33} \end{bmatrix}, \begin{bmatrix} F_{11} & F_{12} \\ F_{21} & F_{22} \end{bmatrix} \right)$$

$$O_{11} = F_{11}X_{11} + F_{12}X_{12} + F_{21}X_{21} + F_{22}X_{22}$$
$$O_{12} = F_{11}X_{12} + F_{12}X_{13} + F_{21}X_{22} + F_{22}X_{23}$$
$$O_{21} = F_{11}X_{21} + F_{12}X_{22} + F_{21}X_{31} + F_{22}X_{32}$$
$$O_{22} = F_{11}X_{22} + F_{12}X_{23} + F_{21}X_{32} + F_{22}X_{33}$$

Figure 9 : Forward Propogation Equations, where F is the filter matrix that sides over the image matrix X giving an activation map O.

1. Similarly as seen in figure 9, multiple filters create a series of activation maps, r, which is again used as input for next level of CNN layers.
2. Along with CNN layers we also have pooling layers.

- **Pooling Layers :**

  Its function is to progressively reduce the spatial size of the representation to reduce the number of parameters (Figure 9) and computation in the network, and hence to also control overfitting.

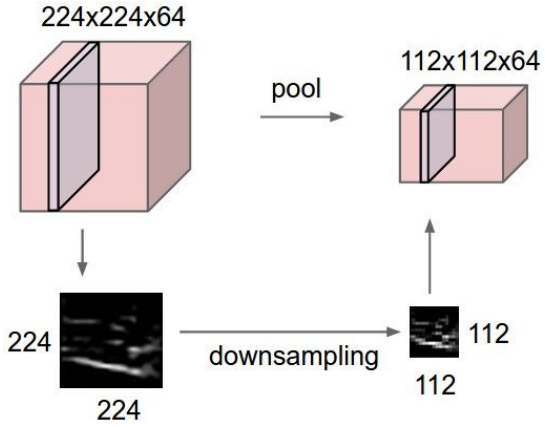


Figure 9: Pooling Layers

Thus convolutional layers increase the length of the input matrix and pooling decreases the height and width. There are mutliple techniques for pooling to be done for example max pooling reduces the size by taking only the maximum pixel value present in a filter and global average pooling take the average of the whole matrix.

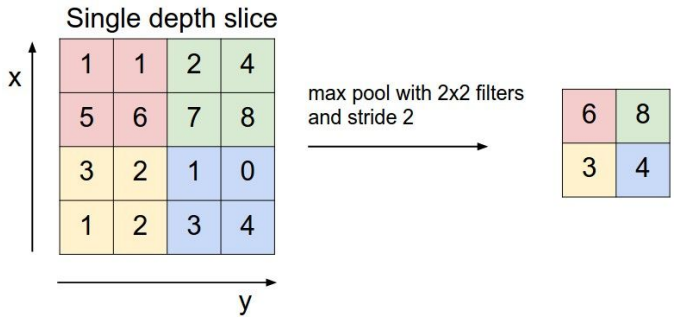


Figure 10: Example of Max Pooling Layers

- **Activation Layers :**

  This is another type of layer that is used along with pooling layers and convolutional layers while training the network. When working with data it is not necessary that they possess a linear relation among themselves, therefore, to introduce non-linearity Activation layers are used.

  Activation function decides whether a neuron should be activated or not by calculating the weighted sum and further adding bias with it. The purpose of the activation function is to introduce non-linearity into the output of a neuron.

- **Back Propagation :**

  When the ouput is generated by forward propagation it is then compared with the actual label, and they both are used to calculate the error function. The Backpropagation algorithm looks for the minimum value of the error function in weight space using a technique called gradient descent. The weights that minimize the error function is then considered to be a solution to the learning problem.

  According to the error function backpropagation is done and the error is distributed among different weights, and accordingly, weights are changed to provide the most optimal solution. Mathematics behind backpropagation can be found at : https://bigtheta.io/2016/02/27/the-math-behind-backpropagation.html

- **Optimizers :**

  During the training process, we tweak and change the parameters (weights) of our model to try and minimize that loss function, and make our predictions as correct as possible. This is where optimizers come in. They tie together the loss function and model parameters by updating the model in response to the output of the loss function. In simpler terms, optimizers shape and mold your model into its most accurate possible form by futzing with the weights. The loss function is the guide to the terrain, telling the optimizer when it's moving in the right or wrong direction. There are multiple optimizers

with their own properties like SGD, Adadelta etc. A detailed view of different optimizers can be found at https://keras.io/optimizers/ .

Since our data consist of images of tissues, therefore, CNN would be the best choice, as it would retain the 2D structure of an image thus allowing the network to learn spacial features as well. CNN understands the fact that the image pixels that are closer in proximity to each other are more heavily related than they are apart therefore it will also have lot less parameters to train hence also allowing faster and memory efficient way for training.

- **Transfer Learning**

  Another concept that I have used while training is transfer learning. When CNN learns a data, the top layers of the model learns more visual characters such as shape, size etc as we go deeper it learns more complex parts of the data. So if we have a dataset completely unrelated to what earlier model is trained for, the initial layers will still be able to identify the shapes and sizes of the images present in this data as they are top layers are not yet specific to the data until they go deeper. Therefore the idea is to use another model like VGG-16 and VGG19 etc that have been trained on a much larger dataset know as Imagesnet. We take the initial layers from other models and add some of our own layers to make it data specific to our domain. We train only the layers that we added as initial layers can already identify visual parts as they did in their original dataset. Thus using the knowledge of other models to train a network on your data is known as transfer learning. It saves a lot of time for training all the layers everytime from scratch.

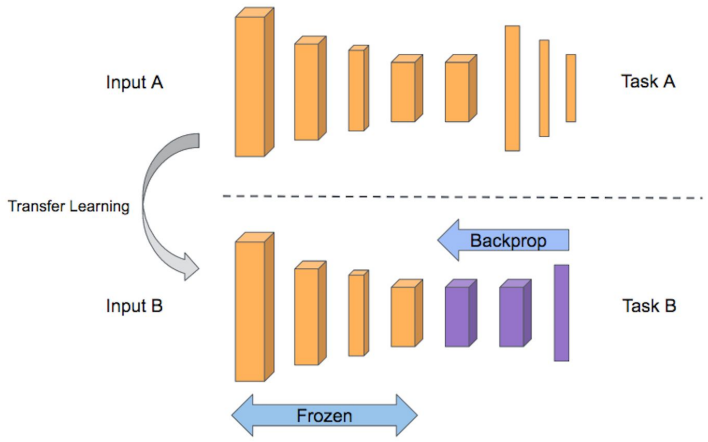


Figure 11 : Showing transfer Learning

Following are the number of parameters that can be tuned to optimize the classifier :

- Training Parameters
  - Number of Epochs

    - Batch Size(number of images to take at once for a single training step)
    - Learning Rate(How fast algorithm should learn)
    - Weight Decay(Prevents neuron domination)
    - Momentum(previous steps are kept in mind while learning further)

- Neural Network Architecture
    - Number of layers
    - Layer Types
        - Convolutional Neural Network
        - Fully Connected
        - Max Pooling

- Optimizers (They are used to decide how the weights have to be updated along with learning rate , momentum etc.)
- Preprocessing Parameters

Click the links to know more about them

Training data and validation data are loaded into the RAM while training, then random batches are selected from the set and loaded on GPU to train the network and calculate validation loss and accuracy. The best epoch weights are stored in an external file which is then later loaded. This prevents overfitting of our network on the data.

Weakness

As we have already talked about the strengths of CNN's there are some drawbacks too such as they are :

- Computationally expensive.

  Like other neural networks, they require a lot of time to get trained. Luckily this was some tradeoff that we could do.
- They require large amount of data

  If you provide them with less, expect the CNNs to perform poorly. CNNs have millions of parameters and with small dataset, would run into an over-fitting problem because they needs massive amount of data to quench the thirst. But our data is large enough to train a CNN, therefore we do not have such a problem.

Benchmark

To create initial benchmark I chose a vanilla CNN architecture which looked like :

```
model = Sequential()
model.add(Conv2D(filters=32,kernel_size=(3,3),strides=2,padding='same',activation='relu',inp
ut_shape=(50,50,3)))
model.add(Flatten())
model.add(Dense(2, activation='softmax'))
model.summary()
```

A single layer of CNN was used containing 32 filters, kernel size of (3 X 3) , strides of 2 and activation *Relu.*
Which is then followed by flatten() layer which converts the output of CNN into a vector which is then passed through a dense layer with activation *softmax.* This will give the final prediction probabilities of the classes.

Model Summary:

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_2 (Conv2D)            (None, 25, 25, 32)        896
_________________________________________________________________
flatten_2 (Flatten)          (None, 20000)             0
_________________________________________________________________
dense_63 (Dense)             (None, 2)                 40002
=================================================================
Total params: 40,898
Trainable params: 40,898
Non-trainable params: 0
_________________________________________________________________
```

There are total of 40,898 parameters that are trained in our benchmark model.
This was the architecture that was used as the benchmark model, as it is the most basic architecture of any cnn model, the results obtained by it would be the the most basic one.
Training and testing of this model and the final model is done on the same data, therefore clear

objectives can be made whether or not the final architecture is learning better the basic one otherwise the final model is poor to use.

Methodology

Data Extraction

Our data involves images which are all given in a good format. But we don't have their respective classes with us, they are embedded in the filenames therefore was needed to be extracted. Last 5 letters of the filename represents the class it belonged. After extraction the data was saved at output.csv. This would serve as our target label. It was also required to shuffle the data so that there are not present in any order.

Data Pre-Processing

Following are the processing steps that were required with our data:

1. Hot encoding of our target labels.
2. Load the images and resize them.
3. Split our dataset into train, test and valid.
4. Reducing the pixel range of images to 0-1
5. UnderSampling
6. Image Argumentation

1. **Hot encoding of our target labels**

Our target labels are in the form class0 and class1 representing IDC(-) and IDC(+) respectively. We hot encoding was done on them to make it work with our models. The hot encoding was done using *keras.utils.to_categorical*.

2. **Load the images from the link and resize them**

The images were of various sizes and for our model to work best we need all images of the same sizes, it eases the learning part and in order to pass the information present in our images we would need to pass the matrix of pixels therefore the images were loaded using opencv and were resized to 50 X 50 .

3. **Split our dataset into train, test and valid**

We had a single data with images files, we need to separate them in various sets. We require one set to train our model, another set to test them and one more set of validation to avoid overfitting or underfitting. Since our data was uneven there while using sklearn `train_test_split` we would also do some undersampling. And also since the data is large, it was best to take a part of it for the project as considering the limited GPU and Ram size. These would be compensated with the undersampling and image argumentation techniques.

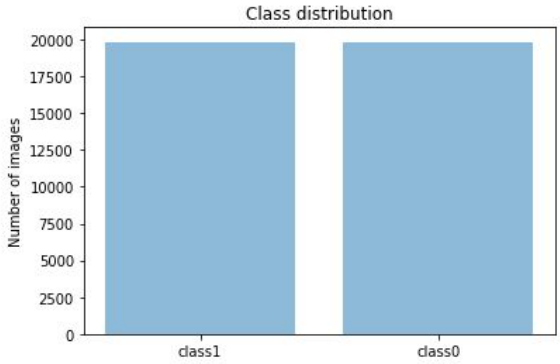


Figure 12 : Class distribution after performing undersampling and data reduction.

First data was split into training set and testing set , then testing set was split into testing and validation set.

Testing set is 30% of the total data and validation set is 20% of the testing set.

Number of train files 39702
Number of valid files 3382
Number of train_target files 39702
Number of valid_target files 3382
Number of test files 13528
Number of test_target files 13528

Files distributions

Training Data Shape: (39702, 50, 50, 3)
Validation Data Shape: (3382, 50, 50, 3)
Testing Data Shape: (13528, 50, 50, 3)
Training Label Data Shape: (39702, 2)

Validation Label Data Shape: (3382, 2)
Testing Label Data Shape: (13528, 2)

Files distribution and shapes

4. **Reducing the pixel range of images to 0-1**

Pixel range involved in our images ranges from 0-255, with most colors residing on the brighter side. We need to lower down the range to 0-1. This makes the model works better with the data. Therefore all the three sets of data was divided by 255.0.

5. **Under Sampling**

Undersampling techniques were used to treat the class imbalance present in our data. Undersampling reduces the the data from the class which is larger than the other. After performing it we have the following class distributions :

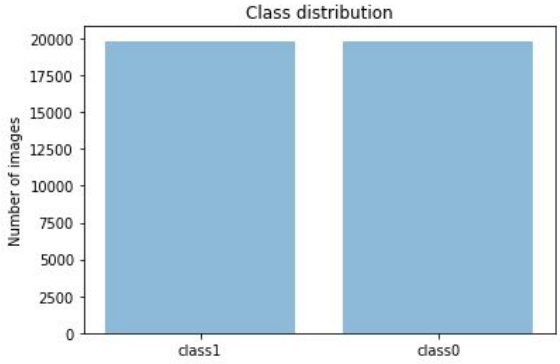


Figure 13: Class distribution after performing undersampling and data reduction.

6. **Image Argumentation**

Since we have reduced our data image argumentation could help in compensate that. It adds more data to the network by rotation , shifts, flips etc of the original image. It can also help in better understanding the images present in our data.

```
shear_range=0.2, (Shear Intensity (Shear angle in counter-clockwise direction in degrees))
rotation_range=40, (Degree range for random rotations.)
width_shift_range=0.2,
height_shift_range=0.2,
zoom_range=0.2,
rescale=1/255.0,
horizontal_flip=True, (Horizontally flip the images)
vertical_flip=True) (Vertically flip the images)
```

Above mentioned parameters were changed and used, although there are far more parameters in Image argumentation techniques which can be used further they are mentioned in : Image argumentation.

Implementation

The implementation is done in two parts first using Image Argumentation and second using transfer learning technique.

Image Argumentation

Steps that were used were :

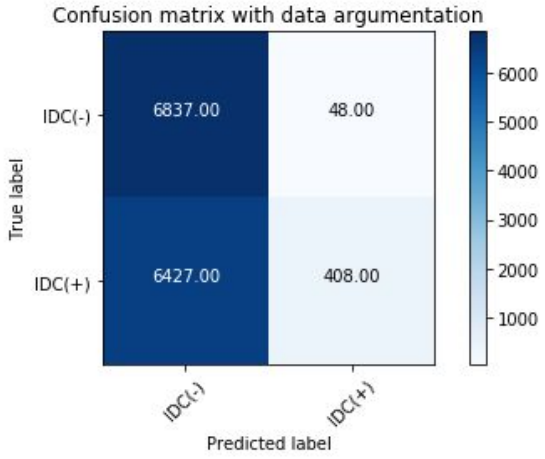
1. The images in the training set were augmented with features that were mentioned in preprocessing set.
2. The valid set and train set in rescaled to 0-1.
3. Then the training set was passed into the architecture for training and validation set was used to check overfitting.
4. A ***checkpointer*** was created for loading the best weights while training.

5. The model was trained on 20 epochs with batch_size= 32, and 5 epochs with batch_size=64, the best epochs were saved in the file named 'weights.bestarg.hdf5'
6. The saved file was then loaded and were used as final weights.
7. Then predictions were made on the test set and to evaluate our model we have used confusion matrix.

When working with image argumentation model, I started with the benchmark model with the only difference that this time I used image argumentation technique. The model was a single layer with the number of filters equal to 32, kernel size of (3 X 3) , strides of 2 and activation *Relu.*

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_2 (Conv2D)            (None, 25, 25, 32)        896
_________________________________________________________________
flatten_2 (Flatten)          (None, 20000)             0
_________________________________________________________________
dense_2 (Dense)              (None, 2)                 40002
=================================================================
Total params: 40,898
Trainable params: 40,898
Non-trainable params: 0
_________________________________________________________________
```

However the result I received was better than the benchmark model, the model was not giving the same prediction for all. The confusion matrix for the initial model was:



Though my model was now predicting both the classes unlike the earlier versions of it, still the model was too shallow, therefore increasing the number of layers and several tuning of parameters, helped me achieved the result I was looking for.

Transfer Learning

Steps that were used were :

1. Data that has to be used for transfer learning was loaded without the top layer.
2. Then all the three sets of train, valid and test were preprocessed and their bottleneck features were calculated.
3. The idea was to hold the bottom layers weights and only train the top layers.
4. The bottleneck_train that was calculated was used to train and bottleneck_valid to validate .
5. A ***checkpointer*** was created for loading the best weights while training.
6. The model was trained on 5 epochs with batch_size= 32, the best epochs were saved in the file named 'weights.bestarg.tranfer.hdf5'
7. The saved file was then loaded and were used as final weights.
8. Then predictions were made on the test set and to evaluate our model we have used confusion matrix.

Complications

There were some major complications while working on this dataset.

- Class Imbalance
- Large data relative to available RAM
- Large data created difficulty in argumenting data directly.
- There was a certain order while loading the data i.e the class0 occurred at once then class1 images, therefore model learn inaccurately and predicted the same value for 0 for all images.

Refinement

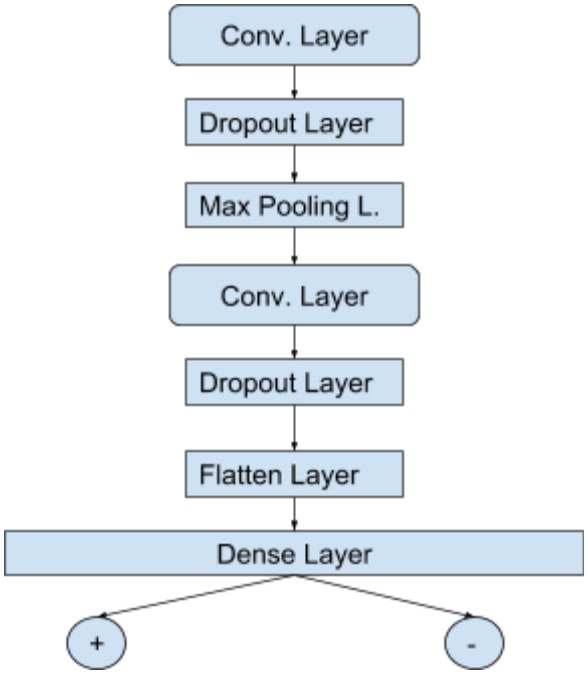
All the complications were resolved by reducing the data size and handling class imbalances with undersampling as mentioned in preprocessing steps. The data was shuffled using *sklearn.utils.shuffle* for removing the order from the data. In intermediate step I also tried argumenting data batch by batch but after reducing the size, the data was easily augmented while training.

Image Argumentation Model

Initially I started with the benchmark model, started adding layers and increasing filter sizes, later on I saw some sought of overfitting, therefore I added dropout layers , with probabilities in propositional to the number of filters in the previous layers and then I saw that model was not learning after the first epoch therefore addition of more layers and changing the optimizer from adam to Adadelta resolved my issue. Also tried various values for various parameters

present in image argumentation. But my final selection of parameters were as mention in preprocessing step and
The final architecture used was :

Model Architecture



Loss that was used was ***categorical crossentropy ,*** optimizer used was ***Adadelta.***

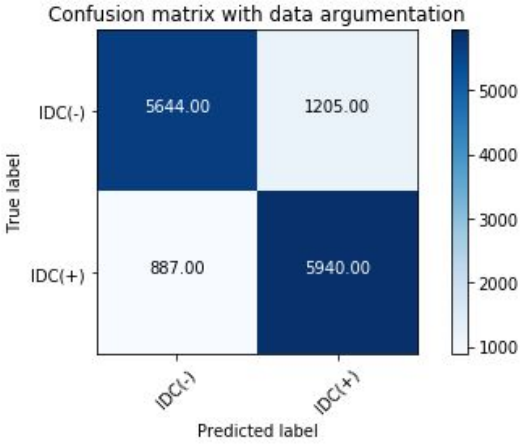


Figure 8 : The diagonals are darker than the non diagonal regions, hence it shows more correct predictions were done, than the wrong ones.

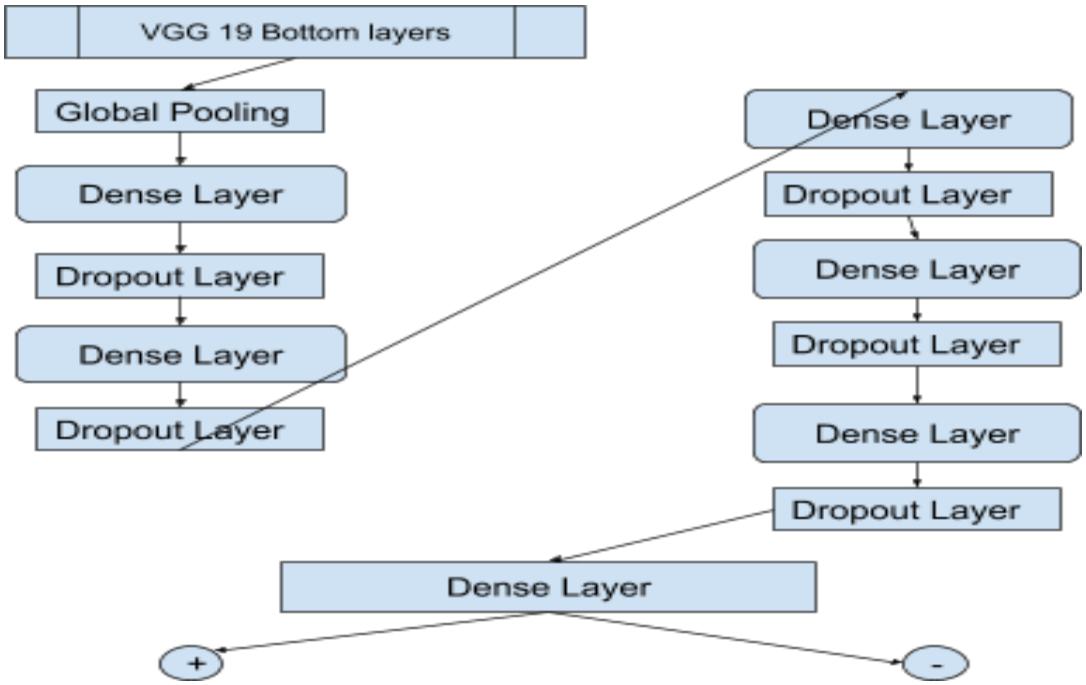
Sensitivity = 0.87	Specificity = 0.82

Transfer Learning

With transfer learning, bottleneck features were need to be calculated, first I tried using Inception model but the small size of our images were not compatible with the model. Therefore tried VGG16 and VGG19 models. VGG19 were found to be the best one.

VGG19 bottom level weights were put on hold and top level were created and trained. Various dense layers were added to better understand the data and dropout layers to avoid overfitting. The filters in each layers were increased as we go deeper so that it can learn more and more as the images traverses through the layers. Dropout probabilities were kept in proportional to the layers. Slowly increased the layers every time I trained my model but the best architecture I found was:

Model Architecture



Loss that was used was ***categorical crossentropy ,*** optimizer used was ***Adadelta.*** They were kept same as the previous model as they gave the best results. The final confusion matrix for this model is :

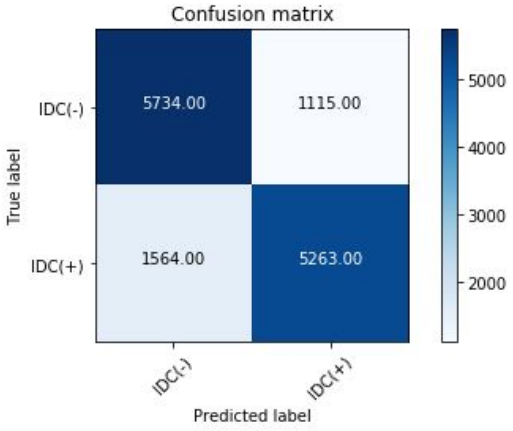


Figure 9 : The diagonals are darker than the non diagonal regions, hence it shows more correct predictions were done, than the wrong ones.

Sensitivity = 0.77	**Specificity = 0.83**

Results

Model Evaluation and Validation

During training the validation set were used to validate the data and avoid overfitting. There are two models that I created first using image argumentation and second using transfer learning. The both were chosen because there were working best with available combination of parameters. I finally chose the image argumentation model as my final model as it produces better sensitivity and specificity.

The test data that are completely unseen to the model were used to measure the sensitivity and specificity of the model and it worked great with them.

The final model :

- The shape of filters of convolutional layers is 3X3.
- The first layer had 32, second layer had 64, then 128 then 512.
- The convolutional layer had stride of 2, so the resolution of output matrix is half the input.
- There is a single max pooling layer that halves the resolution too.

- Dropout layer was added to avoid particular neuron domination.
- Flatten layer is added to convert the matrix into vector for fully connected layer.
- Finally Dense layer is added for classification, at the end softmax is used for probabilistic output for each class.
- Activation function used was relu at the end of each layer
- Loss was categorical cross entropy.
- Optimizers used was Adadelta

To verify robustness of the model 5-fold validation is done on the data and for every fold it was found that the validation score is within 0.01 that is very small and thus acceptable and therefore proves the robustness of the model.

```
array([0.85985, 0.8545 , 0.84025, 0.84905, 0.85525])
```

Justification

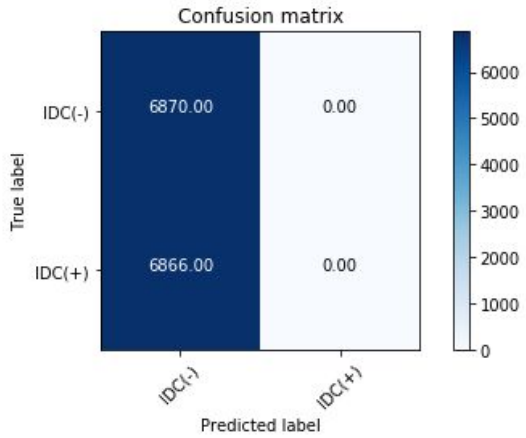


Figure 14 : `Benchmark Confusion matrix`

It can be seen in the benchmark confusion matrix that the model gave the same predictions for all the images. Our final Image argumentation model gave far better results, than the benchmark model. It can be also been seen by comparing the results (sensitivity and specificity) in the following figures.

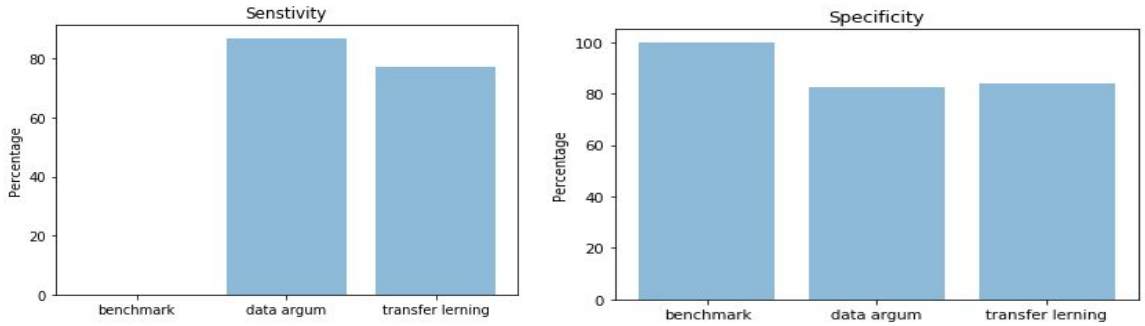


Figure 15 : It can be clearly seen that data argumentation model and transfer learning model works far better than our benchmark model.

The final sensitivity achieved is good enough to solve the problem. The machine can be automated with such models to work alongside human and provide greater breast cancer detection efficiency and reduce time.

Conclusion

Free-Form Visualization

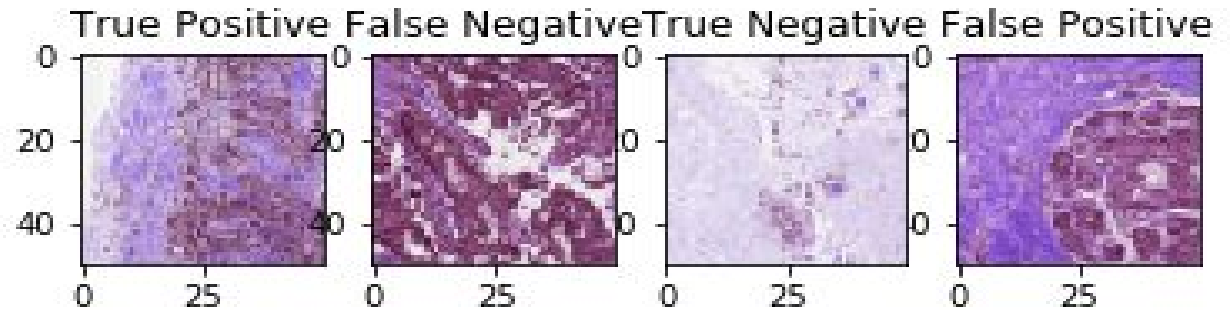


Figure 16 : ***Images of tissues with the prediction done on them versus actual result***

In the above figure all the four possibilities of images have been shown.

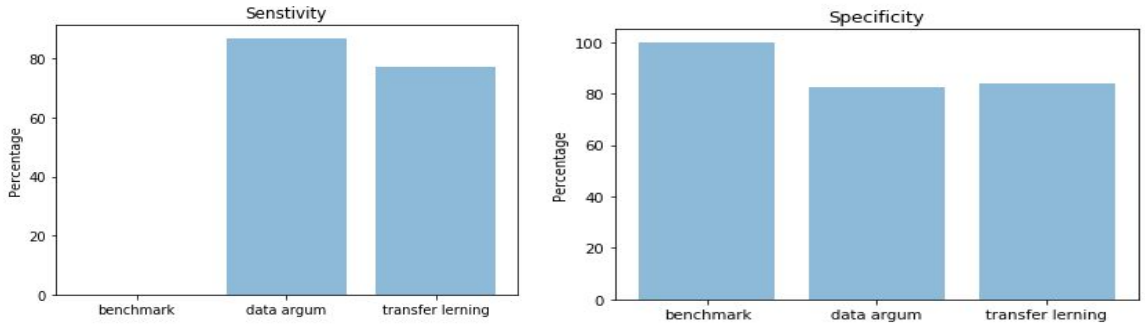


Figure 17

Figure 17 shows that sensitivity achieved by data argumentation model is more than the benchmark and transfer learning model. The high sensitivity means that the model is able to correctly classify breast cancer being positive for the most of the images correctly.

Algorithm Responses

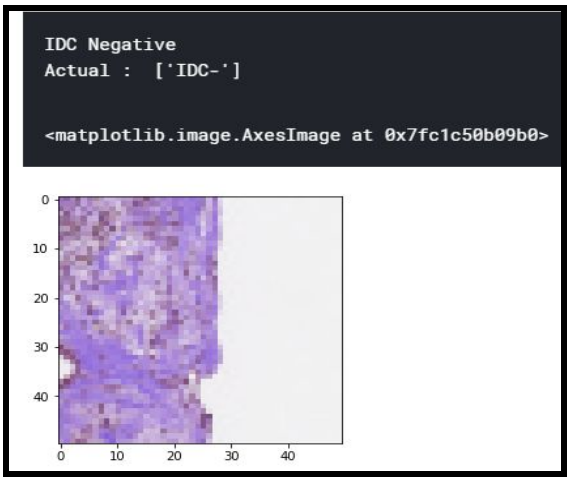


Figure 18

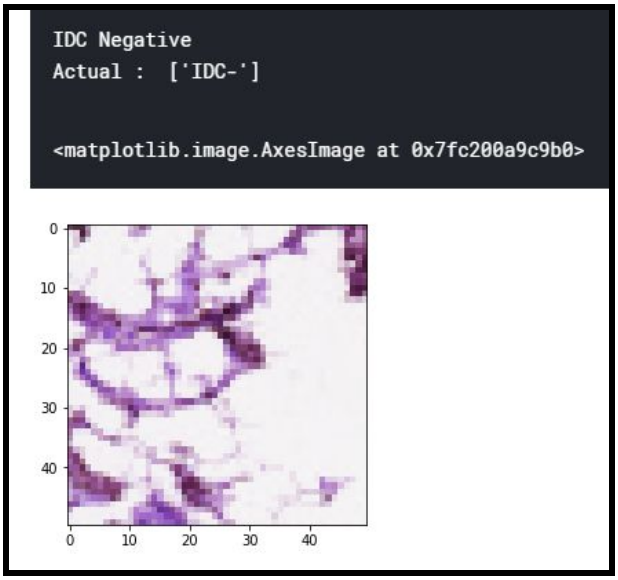


Figure 19

Reflections

The summation of all the process are as follow:

1. The project started with analyzing the dataset, it was found that the images are too small to read, there either 50X50 dimensions or less.
2. Next it was found out that the labels of each images are present in the filenames therefore they were extracted and stored in output.csv.
3. Dimensions of all the images should be equal and preferably in square to work best with our models therefore all images were converted into 50X50 dimensions.
4. Since the images were of tissue therefore they were not converted into grayscale as color can also be a good aspect for predicting.
5. Our labels are stored in output.csv in categorical form, for them to better work with our data the need to be hot encoded. Therefore scikit learn to_categorical were used to do so.
6. The data provided were very large to deal with the available GPU and RAM, therefore the data was reduced for providing better debugging.
7. The data was distributed into train set for training, validation set to validate and avoid overfitting and test set to test the model for completely unseen data.
8. There were class imbalance which need to be taken care of therefore Undersampling was used to handle them.

9. It was also found that there is certain pattern in labels, all class0 were present at first then class1 labels therefore data was shuffled to remove it.
10. After this a benchmark model was created , the final model should surpass this.
11. Benchmark model had only a single layer with CNN layer.
12. Benchmark model was trained and validated , the best weights stored and confusion matrix was drawn.
13. It was found that the model is giving same prediction for all images.
14. Later the image argumentation was done on images, to increase our dataset and for allowing the model to better learn the data.
15. The model was too shallow to learn, the predictions were still same for all images therefore model was rebuilt .
16. The model thus created had 4 convolutional layers which were succeeded by dropout layer to prevent neuraon domination.
17. The loss function used was categorical crossentropy and optimizer that best suited was Adadelta.
18. The sensitivity and specificity along with confusion matrix was calculated to check the final solution.
19. After which transfer learning method were also used to check any improvement that can be done.
20. Transfer learning model that were taken wer VGG16 and VGG 19 . VGG 19 were found to be the best to work with the images.
21. The VGG19 model was put on hold in the bottom part and top part were changed with the layer i designed .
22. The bottleneck features were calculated and the top part was trained and validated.
23. The final solution i.e sensitivity and specificity calculated were better than benchmark model.
24. Since we are working with medical images, sensitivity is important aspect and Image argumentation model worked better than the transfer learning model.
25. Therefore image argumentation model was used as a final model.
26. The comparisons were visualized to better understand them.
27. The algorithm was created to work with unseen data.
28. The final model achieved sensitivity of 0.87 that far better than benchmark and is also a good result considering the medical images.

Improvement

There are certain improvements that can be done such as :

- Applying image argumentation and transfer learning together to work with the data. This could improve Sensitivity.
- If better RAM and Gpu are made available , the whole data can be used and checked if it improves learning.
- Very few positive data was present, if it could be increased the model can learn in a better way.

References

- ➢ **https://medium.com/infosimples/understanding-convnets-cnn-712f2afe4dd3**
- ➢ **https://medium.com/@phidaouss/convolutional-neural-networks-cnn-or-convnets-d7c688b0a207**
- ➢ https://medium.com/@2017csm1006/forward-and-backpropagation-in-convolutional-neural-network-4dfa96d7b37e
- ➢ http://cs231n.github.io/convolutional-networks/#pool
- ➢ https://www.edureka.co/blog/backpropagation/
- ➢ https://blog.algorithmia.com/introduction-to-optimizers/
- ➢ https://www.researchgate.net/post/What_are_Convolutional_Neural_Networks_CNN_weakness

---