A Mini- Project Report on

“Breast Cancer Detection”

[Code No: Comp 484]

For the partial fulfillment of Fourth Year/ First Semester in Computer Engineering

Submitted by:

Kishan Jaiswal(21)

Jasmin Karki(23)

Anukul Parajuli(34)

Divash Ranabhat(42)

Submitted to:

Dr.Bal Krishna Bal

Associate Professor

Head of Department

Department of Computer Science and Engineering

Kathmandu University, Dhulikhel

Kavre, Nepal

Date of Submission:

11 March, 2020

Table of Contents

1. Introduction

Breast cancer is a one of the most dangerous and prevalent causes of death in women that has spread worldwide in current scenario. The oldest documented case of breast cancer was in Egypt in 3000 BC. Breast tumor is an abnormal growth of tissues in the breast, and it may be felt as a lump or nipple discharge or change of skin texture around the nipple region. The rapid advancement of technology in machine learning and deep learning continues to lighten the interest in applying these techniques to improve the accuracy of cancer screening. Many imaging techniques have been developed in medical fields for early detection and treatment of breast cancer and to reduce the number of deaths, and many aided breast cancer diagnosis methods have been used to increase the diagnostic accuracy.

Understanding the problem stated we propose a classification model to recognize the probability of breast cancer using CNN. Publicly available dataset from Kaggle was used in classification techniques, training and testing the model.

1.1 Problem Definition

Breast Cancer is the second leading cause of cancer deaths among women. Breast tumors start from the ductal hyper proliferation, and develop into benign tumors or even metastatic carcinomas (1). In the area of medical research, image processing is quite receptive in applications like mammography, computer-aided detection, breast ultrasound and breast MRI. The physician can examine and even diagnose the tumor with the help of the output in terms of an image produced by these techniques. However, stage I breast cancer have higher year survival rates than patients with stage II breast cancer so the early detection is very important (2). The best way to improve the prognosis of breast cancer is early detection. X-ray mammography is the primary method used for early detection of the breast cancer. Although mammography is an effective screening tool, it does not provide certain and reliable results for women with dense breasts as well as in women having surgical interventions. Recent researches have shown that MRI proves to be a better alternative to mammography as it does not involve any radiation exposure. The main drawbacks of breast MRI are: Specificity is too low and interpretation is complex and not standardized, which recommended only the screening of high-risk women. So, a good breast cancer detector is important.

1.2 Motivations for Doing the Project

Among women, breast cancer is the most frequently diagnosed cancer and a leading cause of death. Breast cancer is on the rise across developing nations between 1 in 8. 1 in 12 women will have breast cancer during her lifetime, mainly due to the increase in life expectancy and lifestyle changes such as women having fewer children, as well as hormonal intervention such as post-menopausal hormonal therapy. The World Health Organization (WHO) agencies for cancer research (i.e. International agency for cancer research (IARC) and American Cancer Society) report that 17.1 million new cancer cases are recorded in 2018 worldwide. According to cancer statistics, breast cancer is the second most common and the leading cause of cancer deaths among women, second only to lung cancer. So it is vital to know that the patient has cancer before the condition worsens and cure them in time. Machine learning and deep learning has played an important role in the revolution of medical field and there are many learning techniques to predict and classify breast cancer patterns. In this project we proposed a deep learning CNN model that outputs the risk probability of getting the IDC cancer.

1.3 Objectives of the Project work

- To detect the possibility of breast cancer in the given set of input images.
- To build a CNN model from scratch and compare it with the model created using tensorflow

2. Related works

Breast cancer detection using convolutional neural networks for mammogram imaging system

In this paper, breast cancer detection using convolutional neural network for mammogram imaging system is proposed to classify mammogram image into normal, benign(non-cancerous abnormality) and malignant (cancerous abnormality). Breast Cancer detection Using Convolutional Neural Networks (BCDCNN) is aimed to speed up the diagnosis process by assisting specialist to diagnosis and classification the breast cancer. A series of mammogram images are used to carry out preprocessing to convert a human visual image into a computer visual image and adjust suitable parameter for the CNN classifier. After that, all changed images are assigned into CNN classifier as training source. The CNN classifier will then produce a model to recognize the mammogram image. By comparing BCDCNN method with Mammogram Classification Using Convolutional Neural Networks (MCCNN), BCDCNN has improved the accuracy toward classification on the mammogram images. Thus, the results show that the proposed method has higher accuracy than other existing methods, mass only and all argument have been increased from 0.75 to 0.8585 and 0.608974 to 0.8271 accuracy.

Two-phase deep convolutional neural network for reducing class skewness in histopathological images based breast cancer detection

This work presents a two-phase model to mitigate the class biasness issue while classifying mitotic and non-mitotic nuclei in breast cancer histopathology images through a deep convolutional neural network (CNN). First, nuclei are segmented out using blue ratio and global binary thresholding. In Phase-1 a CNN is then trained on the segmented out 80×80 pixel patches based on a standard dataset. Hard non-mitotic examples are identified and augmented; mitotic examples are oversampled by rotation and flipping; whereas non-mitotic examples are undersampled by blue ratio histogram based k-means clustering. Based on this information from Phase-1, the dataset is modified for Phase-2 in order to reduce the effects of class imbalance.

The proposed CNN architecture and data balancing technique yielded an F-measure of 0.79, and outperformed all the methods relying on specific handcrafted features, as well as those using a combination of handcrafted and CNN-generated features.

Image processing for medical diagnosis using CNN (2003)

Medical diagnosis is one of the most important area in which image processing procedures are usefully applied. Image processing is an important phase in order to improve the accuracy both for diagnosis procedure and for surgical operation. One of these fields is tumor/cancer detection by using Microarray analysis. The research studies in the Cancer Genetics Branch

are mainly involved in a range of experiments including the identification of inherited mutations predisposing family members to malignant melanoma, prostate and breast cancer. In bio-medical field the real-time processing is very important, but often image processing is a quite time-consuming phase. Therefore techniques able to speed up the elaboration play an important rule. From this point of view, in this work a novel approach to image processing has been developed. The new idea is to use the Cellular Neural Networks to investigate on diagnostic images, like: Magnetic Resonance Imaging, Computed Tomography, and fluorescent cDNA microarray images.

3. Datasets

Context:

Invasive Ductal Carcinoma (IDC) is the most common subtype of all breast cancers. To assign an aggressiveness grade to a whole mount sample, pathologists typically focus on the regions which contain the IDC. As a result, one of the common pre-processing steps for automatic aggressiveness grading is to delineate the exact regions of IDC inside of a whole mount slide.

Source:

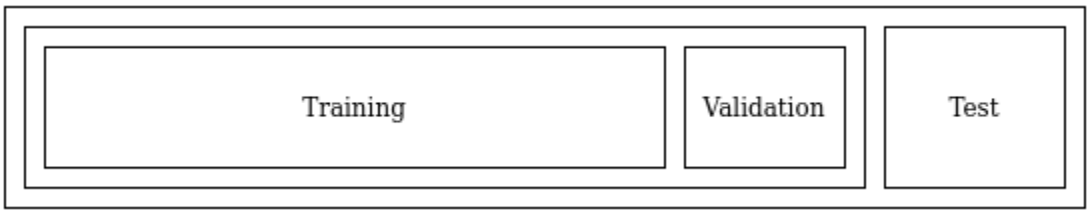
The dataset was downloaded from kaggle.com/paultimothymooney/breast-histopathology-images.

Size and Description of the dataset

The original dataset consisted of 162 whole mount slide images of Breast Cancer (BCa) specimens scanned at 40x. From that, 277,524 patches of size 50 x 50 were extracted (198,738 IDC negative and 78,786 IDC positive). 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. So, the new dataset consists a total of 277,524 images of size 50 X 50 with the file name indicating whether the image contains IDC or not.

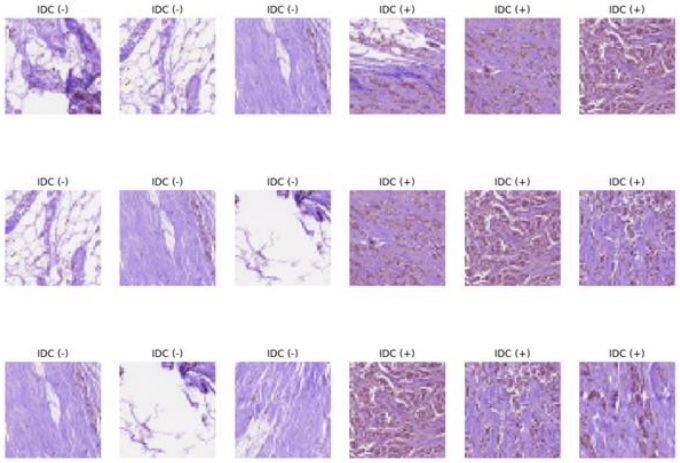
Dataset used for training the model

A small sample of images from the whole dataset of images was chosen to train and test the self-made CNN model. A set of 4320 images (80% of the total data representation) that contained randomly chosen IDC(+) and IDC(-) images were used to train and validate the model, and another set of 1080 images (20% of the total data representation) were used to test the accuracy of the self-made CNN model as well as the CNN model made with deep learning libraries. Only a small sample of the total dataset was taken because of the hardware constraints that we faced.



Visualisation of the dataset

The dataset contains 50*50 (width * height) image patches of IDC(+) and IDC(-) images. Following image shows some examples of both classes of images. Visualizing the data gives us a general idea of the properties of both the class of images. We can see that IDC(+) images are darker with lower pixel intensity values than the IDC(-) images. This feature can be extracted from the images by a CNN upon training.



4. Methods and Algorithms Used

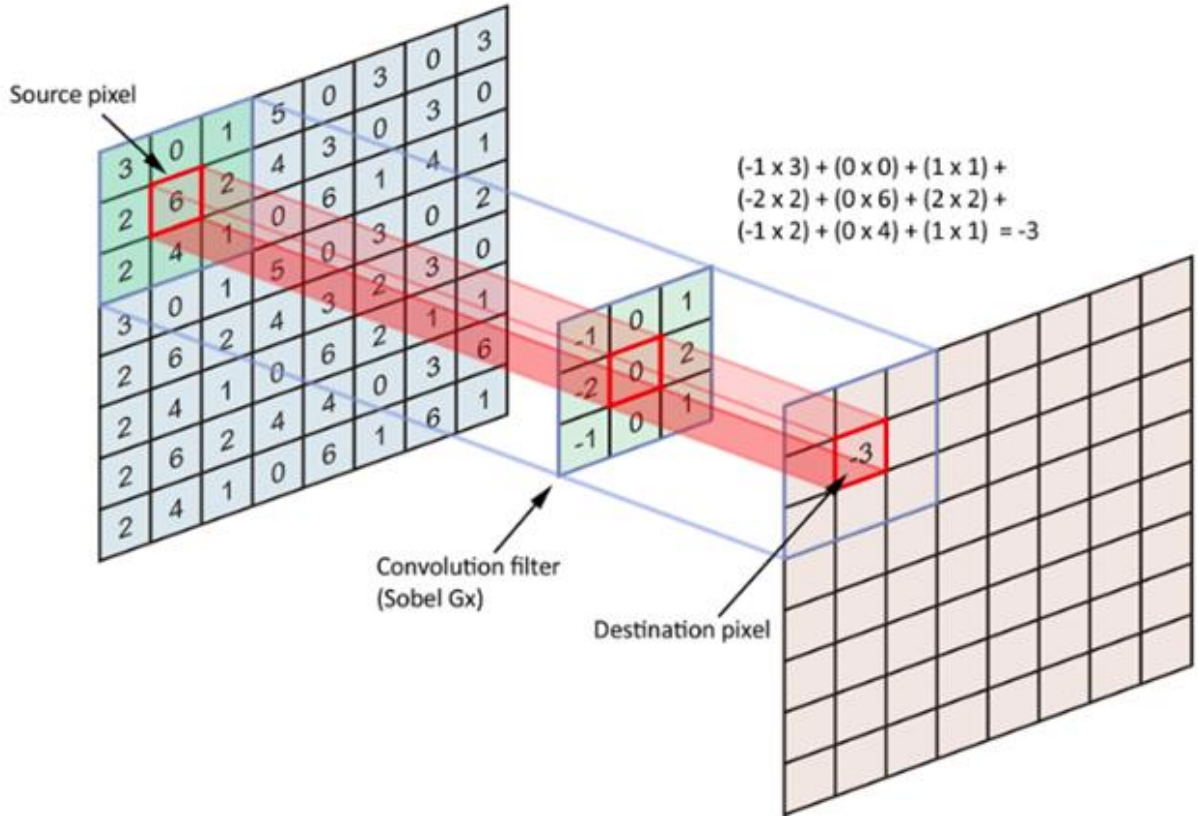
The Machine Learning Algorithm used for the classification of IDC (Invasive Ductal Carcinoma) images was a CNN model. A **Convolutional Neural Network (ConvNet/CNN)** is a Deep Learning algorithm which can take in an input image, assign importance (learnable weights and biases) to various aspects/objects in the image and be able to differentiate one from the other. The pre-processing required in a ConvNet is much lower as compared to other classification algorithms. While in primitive methods filters are hand-engineered, with enough training, ConvNets have the ability to learn these filters/characteristics.
The architecture of a ConvNet is analogous to that of the connectivity pattern of Neurons in the Human Brain and was inspired by the organization of the Visual Cortex. Individual neurons respond to stimuli only in a restricted region of the visual field known as the Receptive Field. A collection of such fields overlap to cover the entire visual area.
Details of a CNN (A deep learning algorithm for image classification)

Convolution Layer (The Filter)
Convolution is one of the main building blocks of a CNN. The term convolution refers to the mathematical combination of two functions to produce a third function. It merges two sets of information. In the case of a CNN, the convolution is performed on the input data with the use of a **filter** or **kernel** (these terms are used interchangeably) to then produce a **feature**

map. We execute a convolution by sliding the filter over the input. At every location, a matrix multiplication is performed and sums the result onto the feature map.
The equation for convolution is given by:

$$G[i,j] = \sum_{u=-k}^{k} \sum_{v=-k}^{k} h[u,v]F[i+u, j+v]$$

Where, G is the output feature map and, h and F are the input and filter applied over the input respectively. The convolution operation can be illustrated more clearly with the following figure:



In the figure given above, the 3x3 filter slides over the 8x8 input and returns its output to the new layer.

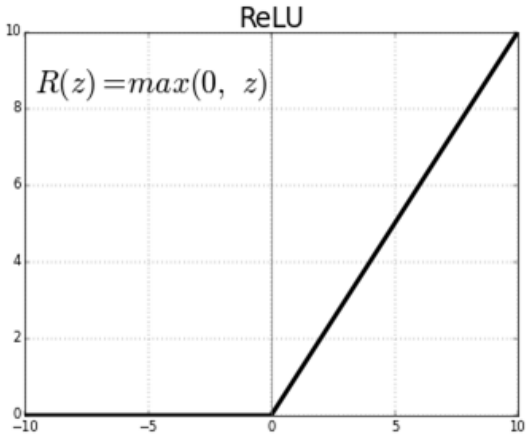
Stride and Padding
Stride is the size of the step the convolution filter moves each time. A stride size is usually 1, meaning the filter slides pixel by pixel. By increasing the stride size, the filter slides over the input with a larger interval and thus has less overlap between the cells.
Padding is the process of adding a layer of zero-value pixels to surround the input with zeros, so that our feature map will not shrink. In addition to keeping the spatial size constant after performing convolution, padding also improves performance and makes sure the kernel and stride size will fit in the input.
The size of the feature map after applying padding (p) and stride (s) is given by:

$$n_{out} = \left\lfloor \frac{n_{in} + 2p - k}{s} \right\rfloor + 1$$

n_{in}: number of input features
n_{out}: number of output features
k: convolution kernel size
p: convolution padding size
s: convolution stride size

Adding non linearity to the Feature map

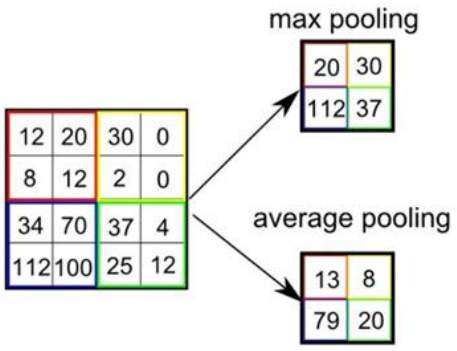
Just like any other Neural Network, we use an **activation function** to make our output non-linear. In the case of a Convolutional Neural Network, the output of the convolution will be passed through the activation function. The following figure shows RelU activation function graph and equation



Pooling Layer

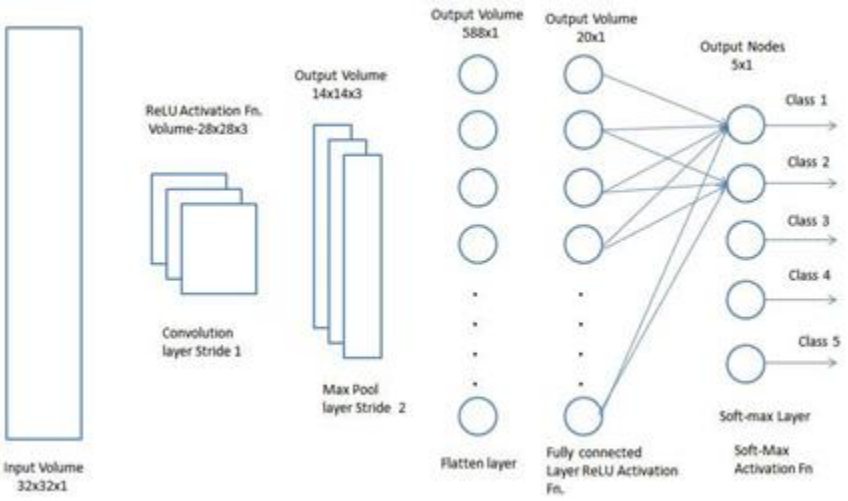
Similar to the Convolutional Layer, the Pooling layer is responsible for reducing the spatial size of the Convolved Feature. This is to **decrease the computational power required to process the data** through dimensionality reduction. Furthermore, it is useful for **extracting dominant features** which are rotational and positional invariant, thus maintaining the process of effectively training of the model.

Max Pooling returns the **maximum value** from the portion of the image covered by the Kernel. Max Pooling also performs as a **Noise Suppressant**. It discards the noisy activations altogether and also performs de-noising along with dimensionality reduction.



In the figure above, we perform max pooling and average pooling with a filter size of 2 and a stride of 2.

Classification – Fully Connected Layer



After the convolution and pooling layers, our classification part consists of a few fully connected layers. However, these fully connected layers can only accept 1 Dimensional data. To convert our 3D data to 1D, we flatten the 3D feature map that we previously obtained from the convolution and pooling layers. This arranges our 3D volume into a 1D vector.

The last layers of a CNN are fully connected layers. Neurons in a fully connected layer have full connections to all the activations in the previous layer. These fully connected layers act as a regular ANN(Artificial Neural Network). These layers also use the RelU activation function for non-linearity.
The last fully connected layer is a SoftMax classifier that produces a probability distribution for the classes of images. These probabilities indicate how well the model has classified the image. The softmax function is given by:

$$f_i(\vec{a}) = \frac{e^{a_i}}{\sum_k e^{a_k}}$$

BackPropagation and Optimization Algorithms

Backpropagation, short for "backward propagation of errors," is an algorithm for supervised learning of artificial neural networks using an optimization algorithm (we have used gradient descent for the example). Given an artificial neural network and an error function (E), the method calculates the gradient of the error function with respect to the neural network's weights. It is a generalization of the delta rule for perceptrons to multilayer feedforward neural networks.

The derivation of the backpropagation algorithm begins by applying the chain rule to the error function partial derivative with respect to a weight given by:

$$\frac{\partial E}{\partial w_{ij}^k} = \frac{\partial E}{\partial a_j^k} \frac{\partial a_j^k}{\partial w_{ij}^k},$$

where a_j^k is the activation (product-sum plus bias) of node j in layer k before it is passed to the nonlinear activation function to generate the output. This decomposition of the partial derivative basically says that the change in the error function due to a weight is a product of the change in the error function E due to the activation a_j^k times the change in the activation a_j^k due to the weight w_{ij}^k.

The first term is usually called the **error**. It is denoted by:

$$\delta_j^k \equiv \frac{\partial E}{\partial a_j^k}.$$

The second term can be calculated from the equation for a_j^k above:

$$\frac{\partial a_j^k}{\partial w_{ij}^k} = \frac{\partial}{\partial w_{ij}^k} \left(\sum_{l=0}^{r_{k-1}} w_{lj}^k o_l^{k-1} \right) = o_i^{k-1}.$$

Thus, the partial derivative of the error function E with respect to a weight w_{ij}^k is:

$$\frac{\partial E}{\partial w_{ij}^k} = \delta_j^k o_i^{k-1}.$$

Thus, the calculated gradient is used to update the weights in any layer of the network using Gradient descent optimization algorithm. The expression for **gradient descent** algorithm is given by:

Repeat until convergence {

$$\theta_j \leftarrow \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta)$$

}

Where, θ maybe any learnable parameter of the CNN (mainly weights and biases).

Calculation of gradients for the proposed CNN Model

By using backpropagation algorithm, the gradients were computed for the weights and biases of each layer. This was necessary for implementing the self-made CNN model. In case of the model built with deep learning libraries, these gradients are calculated and updated by the libraries themselves. The following equations were used to update the weights of the CNN model:

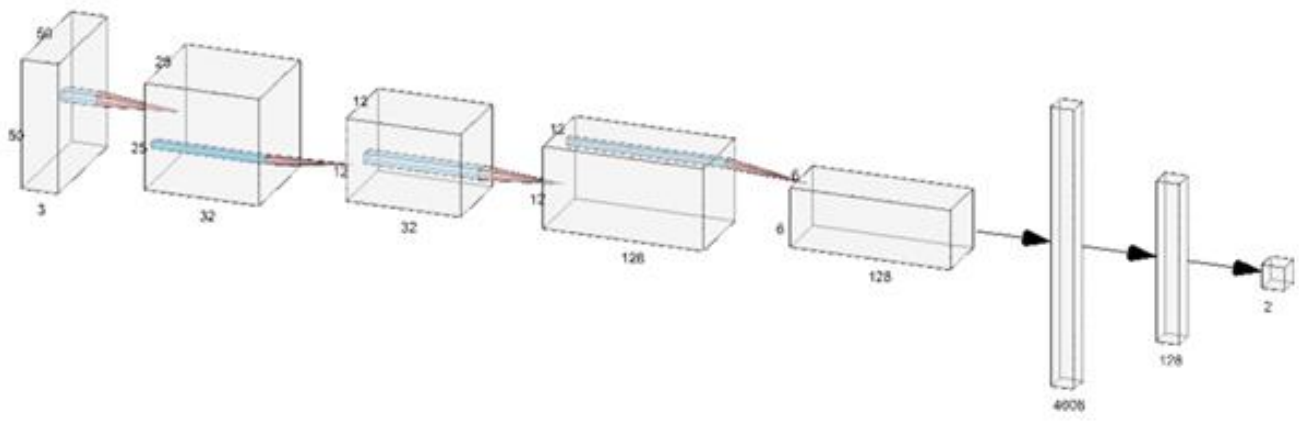
$$\frac{\partial L}{\partial F} = \text{Convolution}\left(\text{Input } X\text{ , Loss gradient } \frac{\partial L}{\partial O}\right)$$

$$\frac{\partial L}{\partial X} = \begin{matrix}\text{Full}\\ \text{Convolution}\end{matrix}\left(\begin{matrix}180^{\circ}\text{rotated}\\ \text{Filter } F\end{matrix}\text{ , }\begin{matrix}\text{Loss}\\ \text{Gradient}\end{matrix}\ \frac{\partial L}{\partial O}\right)$$

The 180 degree rotation indicates that the filter is flipped about the x-axis followed by flipping about the y-axis. The derivatives of the error with respect to the weights and input are both produced by convolution operation between the loss gradient and either the input or the filter.

The Complete CNN Model

The CNN model that we have proposed for the classification of IDC images is given in the following diagram:



From the diagram above, we can see that the parameters are decreasing as we move forward in the model. This is due to dimensionality reduction in CNN.

Justification on choosing CNN model over ANN (Artificial Neural Networks)

The convolution units (as well as pooling units) are especially beneficial as:

- They reduce the number of units in the network (since they are *many-to-one mappings*). This means, there are fewer parameters to learn which reduces the chance of overfitting as the model would be less complex than a fully connected network.
- They consider the context/shared information in the small neighborhoods. This future is very important in many applications such as image, video, text, and speech processing/mining as the neighboring inputs (eg pixels, frames, words, etc) usually carry related information.
- The learned representation of the data is translational invariant. For example, an edge detected in some part of an image can be used to identify similar edge in some

other part of another image. This is not possible in ANN as ANN is not translational invariant.

5. Experiments

The dataset was created as explained above. With the dataset, the model was built by following steps:

1. **Building Multiple Model**
   We built multiple models with different number of filters using Tensorflow. Tensorflow is an open source deep learning library used to build deep learning models. The reason behind building multiple models was due to the low evaluation score of a single filter model. We created the first model with 32 filters, the second model with 128 filters and the third model containing both 32 and 128 filters.

2. **Identifying the Best Model**
   We checked the accuracy of each model with the test set. The accuracy of the third model was found to be the maximum so, we selected it for further evaluation. The evaluation metrics used were precision, recall, f1 score and accuracy which were found to be

3. **Model building from scratch**

   Since the primary objective of the project was to build a CNN model from scratch and compare it with the model created using tensorflow, we built our own CNN model by following steps.

   i. **Normalizing the pixel values**

   The input to the system is 50*50 pixel RGB images. As the pixel values for each image in the dataset are unsigned integers in the range between no color and full color, or 0 and 255, we scaled the images by normalizing the pixel values i.e to the range [0,1].

   ii. **Feature Extraction Convolution**

   We convoluted the input image using 32 3*3*3 filter with stride length 2 and padding of 1. By using the formula ceil[(N-f+2p)/s+1] (where ‘f’ is the filter size and ‘s’ the stride length)

   For first stage, N=50, f=3, s=2, p=1. We get activation shape as (25,25,32) which on 2*2 max pooling with stride value 2 forms a sampled down shape of (12, 12, 32).

   We further convoluted the input image with ReLU activation using 128 3*3*3 filters.

   For second stage, N=12, f=2, s=2, p=1. We get activation shape as (12, 12, 128) which on 2*2 max pooling with stride value 2 forms a sampled down shape of (6, 6, 128).

After going through the above process, we have successfully enabled the model to understand the features. Moving on, we flattened the final output of (6, 6, 128) into a one dimensional column vector of size 4608 to feed into a fully connected layer. Finally, we applied activation function softmax to classify IDC or no IDC with a certain probability distribution.

6. Evaluation of Results

Evaluation of CNN model made from scratch

The results from the CNN model made from scratch and the one made using Tensorflow (a python deep learning library) have been evaluated differently. The self-made CNN model and the one made with Tensorflow is trained on the same data. The evaluation of the CNN model made from scratch has only been done in terms of its accuracy. This is a valid evaluation metric for the model as the dataset used for training the model is a balanced dataset i.e. the number of IDC(+) and IDC(-) images is identical.

Accuracy:

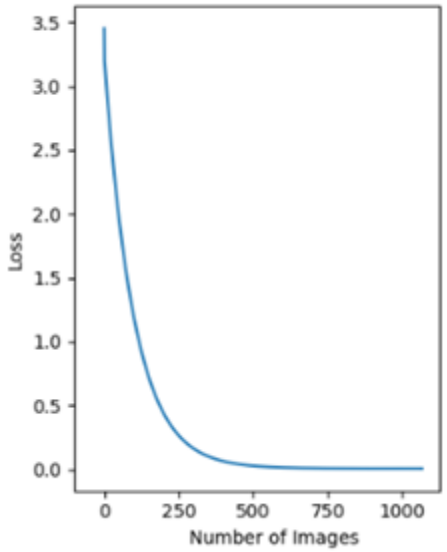
The testing accuracy for the CNN model made from scratch is calculated as :

$$CR = \frac{C}{A}$$

CR – The correct rate;
C – The number of sample recognized correctly;
A – The number of all sample;

The model was able to classify 446 images out of 1080 images tested on the test dataset. So, the accuracy of the model is (446/1080) = 0.4129 i.e. 41.29% accuracy.

The graph for the loss versus number of images trained was obtained as shown below:



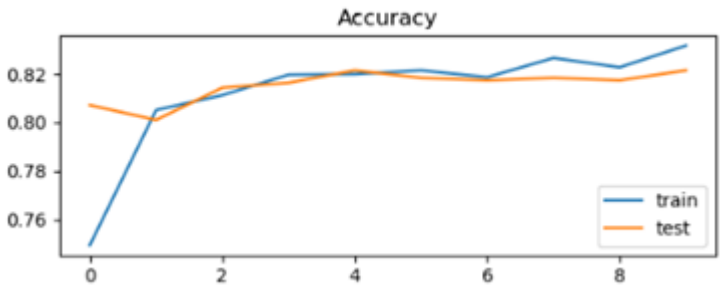
The CNN model used gradient descent for optimizing the weights. From the graph above, we see that the model is clearly overfitting. This is a problem because it may not be able to classify unseen images properly. This may also be the cause of low accuracy (48% accuracy) of the model.

Evaluation of the CNN model made using Tensorflow

The following metrics were used for evaluating the CNN model:

Accuracy

Accuracy is one metric for evaluating classification models. The training accuracy of the model after training on a test data of 3920 (80% of the total dataset used for training) images for 10 epochs increased from a value of 0.74 to 0.84 (74% to 84%). It can be seen from the graph below:



The test accuracy of the model on a validation set of 980 samples (20% of the total dataset used for training) used during training increased from a value of 0.81 to 0.82. The change in the validation accuracy is not very significant. This may be because the model had started correctly classifying images during the initial phase of the training process. Both the training accuracy and validation accuracy are consistently growing and since there is not much difference between these values, the model has not overfit.

The model was tested again using the trained weights saved during the training process. 1080 sample images (this collection of images is a balanced dataset) were used to test the model. 84 percent of the total test images were correctly classified. It means that the model has a test accuracy of 84 percent.

Precision

Precision (also called positive predictive value) is the fraction of relevant instances among the retrieved instances. The precision value for the CNN model was 0.8479 which means that 84.79 percent of the data is relevant.

Recall

Recall refers to the percentage of total relevant results correctly classified by your algorithm. The CNN model had a recall value of 0.8511 which means that it was able to correctly classify 85.11 percent of the total relevant results.

F1 Score

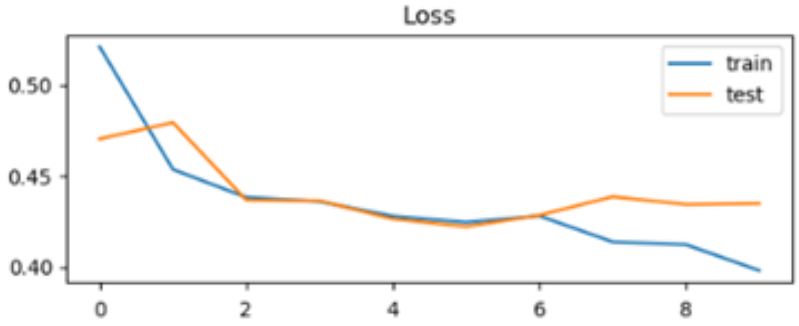
In statistical analysis of binary classification, the **F_1 score** (also **F-score** or **F-measure**) is a measure of a test's accuracy. It considers both the precision *p* and the recall *r* of the test to compute the score: *p* is the number of correct positive results divided by the number of all positive results returned by the classifier, and *r* is the number of correct positive results divided by the number of all relevant samples (all samples that should have been identified as positive). The F1 score for our CNN model was 0.8495.

Loss Function

The loss function for both the CNN model made from scratch and the one made with tensorflow uses binary cross-entropy loss. It is also called log loss and it has been used after the softmax layer. The expression for the loss function is given below:

$$H_p(q) = -\frac{1}{N}\sum_{i=1}^{N} y_i \cdot log(p(y_i)) + (1 - y_i) \cdot log(1 - p(y_i))$$

Here N is the mini batch size, $p(y_i)$ is the probability for the first class of images and $(1- p_i)$ is the probability for the other class. The value y_i is the correct probability mass we feed into the CNN model as labels for the input image. Upon training with a batch size of 10 images for 10 epochs, our CNN model had a significant decrease in loss for both the training and test data.



7. Discussion on Results

The training loss decreased from a value of 0.52 to 0.39 and the validation loss decreased from a value of 0.47 to 0.44. This is not a significant decrease in the loss of validation set which means that training the model with an increased size of training data will not affect the value of validation accuracy and the model will overfit. The ways of improving the model are:

1. For the CNN model made from scratch

The self-made CNN model is complete in terms of the basic functionalities that a CNN should possess but it is still lacking the implementation of mini-batch gradient descent due to which the model is clearly overfitted. It is only able to classify the test images with an accuracy of 41.29% which is still not a good result. The problems that the model currently faces could be:

- The gradient descent algorithm may have fallen into a local maxima due to which the weight updates may not have been optimal. This problem can be reduced by the use of another optimization algorithm called 'adam optimizer'. It solves the problem of getting stuck in a local maxima and saddle points and converges faster than gradient descent algorithm.

$$For\ each\ Parameter\ w^j$$

(j subscript dropped for clarity)

$$\nu_t = \beta_1 * \nu_{t-1} - (1 - \beta_1) * g_t$$

$$s_t = \beta_2 * s_{t-1} - (1 - \beta_2) * g_t^2$$

$$\Delta\omega_t = -\eta \frac{\nu_t}{\sqrt{s_t + \epsilon}} * g_t$$

$$\omega_{t+1} = \omega_t + \Delta\omega_t$$

$\eta : Initial\ Learning\ rate$
$g_t : Gradient\ at\ time\ t\ along\ \omega^j$
$\nu_t : Exponential\ Average\ of\ gradients\ along\ \omega_j$
$s_t : Exponential\ Average\ of\ squares\ of\ gradients\ along\ \omega_j$
$\beta_1, \beta_2 : Hyperparameters$

- The gradients calculated for the weights and biases may not have been optimal as the expressions for the gradients were calculated by ourselves and hardcoded into the model. Also, since the self-made CNN model was trained without a validation set, the learning rate remained constant. So the CNN may not have learned the best representative hypothesis for the training data.
- Application of Batch Normalization may also help to speed up the training process for the CNN model and increase its accuracy. The expressions for batch normalization can be seen from the figure below:

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β
Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma^2_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma^2_{\mathcal{B}} + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma \widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

2. For the model made using tensorflow

All the metrics that were used to evaluate the model shows that the model is moderately accurate. The model could have been called highly accurate if it was able to classify more than 90 percent of the testing data. Following improvements could be made for making a better model using tensorflow.

- The batch normalization technique could also be applied in this model to speed up the training process and increase accuracy by 2 to 4 percent.
- The model could be trained on a larger dataset with the use of computers that have higher processing power to yield better results.

8. Contributions of each group member

Member	Tasks Completed
Jasmin Karki	Using Tensorflow 1. Dataset Creation 2. Multiple Model Creation using Tensorflow 3. Identification of Best Model 4. Evaluation of Model
Divash Ranabhat	Model Creation from Scratch 1. Implemented Convolution 2D 2. Convolution 3D 3. Max Pool
Anukul Parajuli	1. Evaluated Gradients for updating weights 2. Implemented Back Propagation 3. Finding gradients for maxpool layer 4. Dilation Operation
Kishan Jaiswal	1. Trained the CNN model made from scratch 2. Tested the model 3. Evaluated the model

9. Code

Link to the github repository is :

https://github.com/anukulu/Invasive_Ductal_Carcinoma_Detection_Using_Convolutional_Neural_Networks

10. Conclusion and Future Extensions to the Project

The objectives of the project were met but the evaluation metrics showed that the self-made model classified poorly as compared to the model created using tensorflow.

Notable Findings

- The model should be created very carefully, otherwise it will classify poorly.
- The computation complexity of self-made model is very high.
- The model should be trained in different architectures and the best one should be found.

Unfinished Works:

- Training the model with more variation in the model parameters
- Training the model with even more dataset.

References

1. *Risk Factors and Preventions of Breast Cancer.* **Yi-Sheng Sun, Zhao Zhao, Zhang-Nv Yang, Fang Xu, Hang-Jing Lu, Zhi-Yong Zhu, Wen Shi, Jianmin Jiang, Ping-Ping Yao and Han-Ping Zhu.** s.l. : International Journals of Biological Sciences.

2. *The curability of breast cancer: present and future.* **G.N.Hortobagy.** 1, s.l. : European Journal of Cancer Supplements, Vol. 1.