

Principles of Neural Systems
(EGCE 529)

Final Project Report

Prof: Lan Nguyen

BREAST
CANCER
CLASSIFICATION

-Sanjana Lad (885867762)

-Pavan Kumar Kandregula(887319010)

Abstract:

Neural systems, particularly profound learning models such as convolutional neural systems (CNNs), can play a imperative part within the early location of breast cancer. Breast cancer is the driving cause of cancer-related passing in ladies. Early location and exact determination are basic to progressing quiet results. Convolutional neural systems (CNNs) have risen as capable instruments for early location of cancer due to their capacity to memorize and extricate critical highlights from clinical pictures. Utilizing its calculation and convolutional channels, CNNs can distinguish complex designs and inconspicuous anomalies in mammograms, histopathology pictures, and other therapeutic imaging modalities. CNNs, through broad preparing on enormous information, can recognize between delicate tissue and delicate tissue of the breast, enabling exact classification and discovery of tumors within the early days. CNN's layers empower the demonstrate to memorize complex representations, making a difference to distinguish inconspicuous highlights that are not effortlessly unmistakable to a human eyewitness. By giving precise and solid examination of cancer conclusion, CNNs have the potential to bolster radiologists and doctors in demonstrative choices, empower early discovery, moved forward treatment arranging and eventually understanding results. In this extend, we examine the utilize of convolutional neural systems (CNNs) to classify breast cancer utilizing histopathology pictures from the Kaggle dataset. We actualized a CNN show utilizing Python and prepared it on Google Colab utilizing CUDA.

Keywords:

CNN, cancer, python, training

INDEX

Sr. No	Title	Page No.
1.	Introduction	1
2.	Concepts.....	7
	2.1 Gradient Descendent	7
	2.2 Backpropagation	9
	2.3 CNN (Convolution Neural Network)	12
	2.4 Convolution Layers	14
	2.5 Pooling Layer	15
	2.6 Fully Connected Layers	16
3.	Methodology.....	17
4.	Conclusion.....	25
5.	Future work.....	26
6.	References.....	27

1.Introduction:

1.1What are neural networks?

To get it neural associations, we must begin with sensors. Sensors or counterfeit neurons are the scientific structures that make up the brain. Like neurons, sensors acknowledge different twofold inputs to supply a single double yield. The significance of each input can be communicated by including a weight to the input. The neuron's yield is or 1, depending on how much more prominent or less the weight is than the limit. Scientifically, typically:

$$\text{output} = \begin{cases} 0 & \text{if } \sum_j w_j x_j \leq \text{threshold} \\ 1 & \text{if } \sum_j w_j x_j > \text{threshold} \end{cases}$$

Fig 1.1a

where w is the weight of each input and x is the input.

We can get diverse models by changing the weight and edge. Presently, to streamline the way we display the sensor, let's move the edge out of the parameter and supplant it with the so-called sensor inclination $b = -\text{limit}$. Utilizing the inclination from the starting, the perceptron run the show can be composed as:

$$\text{output} = \begin{cases} 0 & \text{if } w \cdot x + b \leq 0 \\ 1 & \text{if } w \cdot x + b > 0 \end{cases}$$

Fig 1.1b

Where W and x are vectors, whose components are weights and input separately.

Now, when these neurons are organized on different levels, it's called a neural organize. A neuron isn't a total show of decision-making, but it outlines how a neuron can weigh up diverse sorts of evidence in arrange to create choices. And it ought to appear conceivable that a complex arrange of neurons could make very inconspicuous choices.

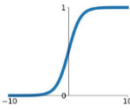
Activation function could be a work that's utilized to induce the yield of a neuron. There are two sorts of actuation capacities: direct and nonlinear (the work over could be a straight actuation work). Nonlinear capacities are most commonly utilized since it makes the demonstrate generalize superior with a wide assortment of information – we are going utilize one in this article. A few of the foremost commonly utilized actuation capacities are:

- 1.Sigmoid function
- 2.Tanh activation function
- 3.Rectified linear Unit or ReLU
- 4.Leaky ReLU

Activation Functions

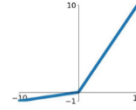
Sigmoid

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



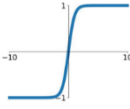
Leaky ReLU

$$\max(0.1x, x)$$



tanh

$$\tanh(x)$$

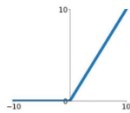


Maxout

$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

ReLU

$$\max(0, x)$$



ELU

$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$

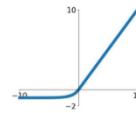


Fig 1.1c

This is what a simple neural network looks like:

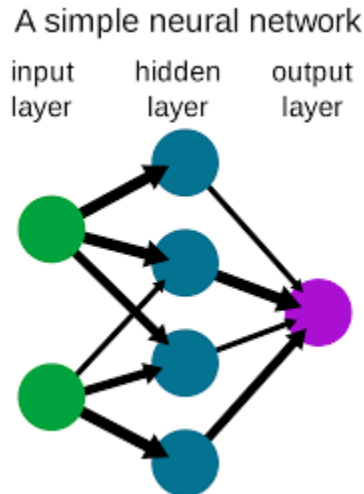


Fig 1.1d

The primary layer is called the input layer and the proper layer is called the yield layer. The layers between these two are called covered up layers. In this organize, the primary set of sensors is decided by measuring the input. The yield is bolstered to the moment layer and so on until the final layer. The complexity of the choice increments with the method, as each sensor makes its choice by weighting the inputs from past forms. In this way, multi-layered sensor systems can be joined into complex choice making. A neural arrange that employments the yield of one layer as input to the following layer is called a feedforward organize.

Presently that we know what neural systems are, let's conversation approximately how they learned to uncover the truth. When we say that the arrange learns, it implies that by calculating or following a few exchanges, the arrange finds the proper weight and predisposition to decrease the loss.

The misfortune is the distinction between the target we get from the w and b values and the predicted product. Our objective is to play down this mistake to induce the foremost exact w and b values. Let's utilize the cruel square mistake work to calculate our mistake.

$$E = \frac{1}{n} \sum_{i=0}^n (y_i - \bar{y}_i)^2$$

Fig 1.1e

Here y_i is the actual value and \bar{y}_i is the predicted value. Let's substitute the value of \bar{y}_i :

$$E = \frac{1}{n} \sum_{i=0}^n (y_i - (wx_i + b))^2$$

Fig 1.1f

Calculating the cruel squared blunder could be a three-step process:

For a given x, discover the contrast between the genuine y-value and the estimated y-value ($y = wx + b$). This can be the square of the distinction. Calculates the normal of the squares of all values in X. So, we square the mistake and discover the cruel. Consequently, the title implies square error. Why is the work not shown? After all, aren't we ordinarily concerned with the number of pictures classified by the organize? Rather than reducing the numeric esteem as an error, why not attempt to do that number directly? The problem is that the number of pictures classified isn't a smooth work of weight and predisposition within the organize. In common, little changes in weights and predispositions don't cause a noteworthy alter within the number of rectify preparing maps. Visit this web journal to studied more. This makes it troublesome to choose how to alter the weights and predispositions to move forward execution. On the off chance that we utilize fetched viability just like the blunder work recorded over, it's simple to decide how to create little changes in weight and drift to move forward cost. Hence, we to begin with center on minimizing the mistake and as it were at that point check the rightness of the classification.

2. Concepts

2.1 Gradient Descent

Presently that we've decided the misfortune work, let's get to the fun portion - minimizing this and finding w and b . Presently, angle plunge calculation is an optimization strategy that finds the least number of operations. Here is the blunder in our work that we said prior. I'll clarify slope plummet in scalar values and after that hop into network operations when talking about image classification since picture may be a framework. Let's attempt utilizing angle plummet for w and b and get one step closer:

1. Initially, let $w = 4$ and $b = 0$. Let our subject be L . This controls how much the esteem of w changes at each step. L can be as little as 0.0001 for great accuracy.

Note that weight w must be initialized as a consistent, not 1 or [more details]

2. Compare the halfway esteem of the misfortune work with w and take the subordinate of D by substituting x , y , w and b .

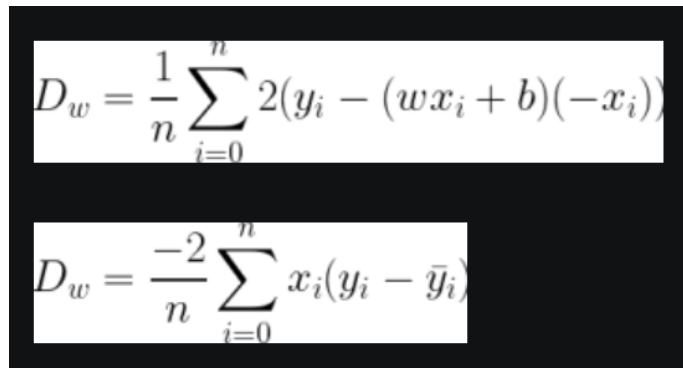

$$D_w = \frac{1}{n} \sum_{i=0}^n 2(y_i - (wx_i + b))(-x_i)$$
$$D_w = \frac{-2}{n} \sum_{i=0}^n x_i(y_i - \bar{y}_i)$$

Fig 2.1a

Now D_w is the value calculated with respect to w . Let's calculate D with respect to b , i.e., D_b .

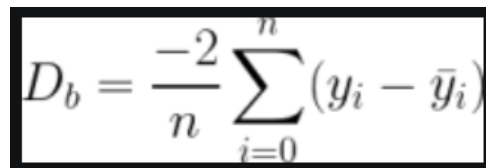

$$D_b = \frac{-2}{n} \sum_{i=0}^n (y_i - \bar{y}_i)$$

Fig 2.1b

3. Now we update the current value of w and b using the following equation:

$$w = w - L * D_w$$

$$b = b - L * D_b$$

4. We repeat this process until our loss function is a very small value or ideally 0 (which means 0 error or 100% accuracy). The value of w and b that we are left with now will be the optimum values. Now with the optimum value of w and b our model is ready to make predictions! Please note that finding the “right set” of optimum values are crucial. Please look in this article to know about overfitting and underfitting of data, which interfere in finding the “right set” of optimum values.

To make gradient descent work correctly, we need to choose a small enough learning rate L so that the above equation is a good approximation, but not too small or the gradient descent will work too slowly. Gradient descent often works extremely well, and in neural networks we’ll find that it’s a powerful way of minimizing the cost function, and helping the net learn.

Now, there’s a challenge in applying gradient descent rules. A quick look at the error function:

$$E = \frac{1}{n} \sum_{i=0}^n (y_i - \bar{y}_i)^2$$

Fig 2.1c

tells us that this is the average of the individual errors of the training model. Actually, to calculate the D gradient, we have to calculate the D_x gradient for each x expression separately and then average them. Unfortunately, this can take a very long time when the training data is very large, so learning can be slow.

Stochastic gradient descent can be used to solve this problem. Here, instead of calculating the actual gradient D , the estimated gradient is calculated for a selected selection from the input or for a small group of small samples. From the mean of this mini-set, we can get a good estimate of the true gradient, which will help ensure gradient descent and learning.

How does this affect learning in neural networks? Let w and b be the weights and trends in our network. Stochastic Gradient Descent works by choosing a small set of training ideas and using them for training. It then selects other groups and uses them for training.

This continues until the training session ends, which is called the completion of the training session. At that time, a new education period began.

It has a fast algorithm for calculating the gradient of the error function called backpropagation.

2.2 Backpropagation

Back propagation refers to how changing the weight and bias in the network changes the error. The purpose of backpropagation is to calculate the partial derivatives Dw and Db of the error function E with each weight w or deviation b in the network.

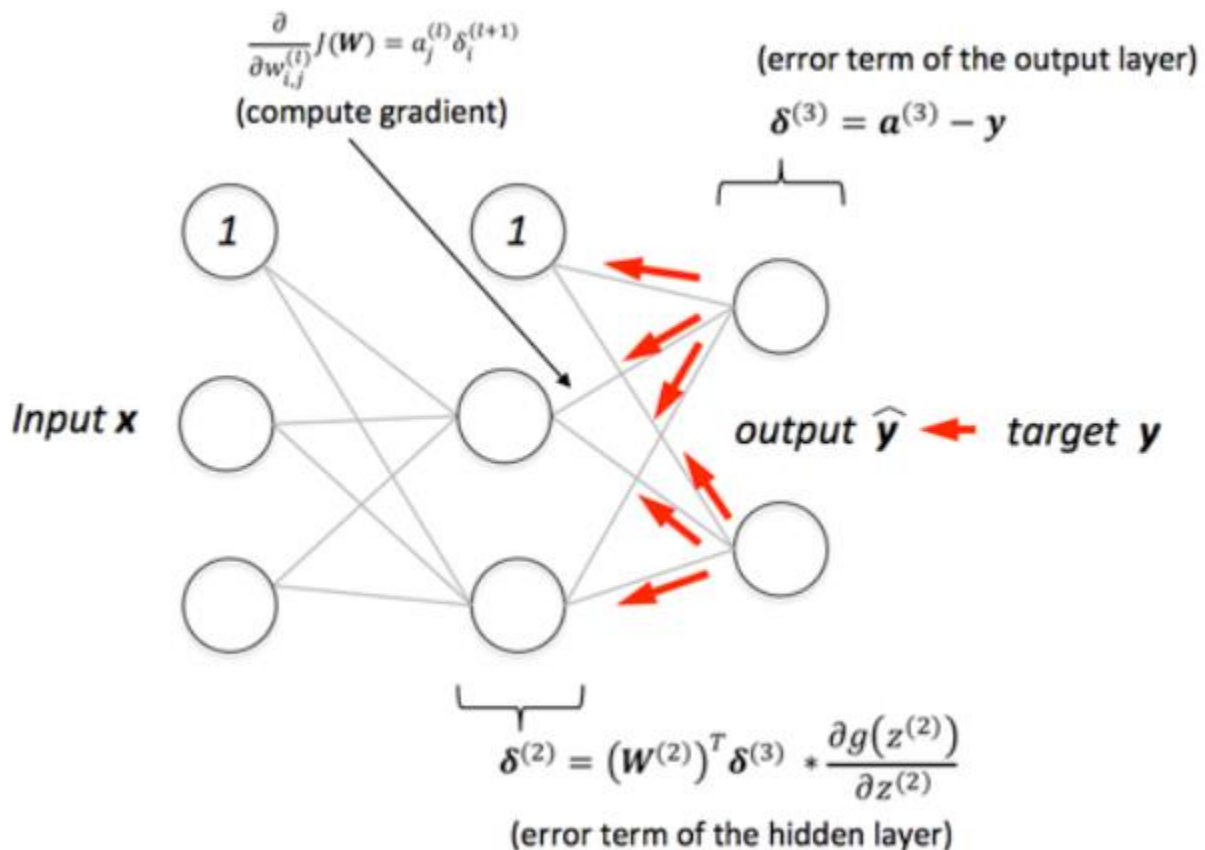


Fig 2.2a

To calculate them, j in layer l . Let me specify the mean value δ_{lj} , which will be the error of the neuron. Backpropagation will then provide a method for calculating δ_{lj} associated with Dw and Db . Let's see how these lines affect our neural network. The error is that layer l is j . is in the neuron. When a neuron's input comes in, the errors interfere with the neuron's activity.

It adds a small variation of Δe_{lj} to the neuron's input weight, so that the neuron outputs y ($e_{lj} + \Delta e_{lj}$) instead of y (e_{lj}). This change propagates through the layers in the network and eventually leads to a change in the total $De_{lj}\Delta e_{lj}$ value.

Back propagation is based on four equations:

1. Output layer error

$$\delta_j^L = \frac{\partial E}{\partial a_j^l} \sigma' e_j^l$$

Fig 2.2b

where E is the error function and σ is the activation function. $\partial E / \partial a_j^l$, j . As the output activation function, it measures how fast the error rate changes. The second term $\sigma' e_j^l$ measures how quickly the function changes in e_j^l . For simplicity, we treat E as a vector and rewrite the above expression (eq 1):

$$\delta^L = (a^L - y) \odot \sigma'(e_L)$$

Fig 2.2c

2. Error in terms of the error in the next layer (eq 2)

$$\delta^L = ((w^{l+1})^T \delta^{l+1}) \odot \sigma'(e_L)$$

Fig 2.2d

where $(w^{l+1})^T$ is the transpose of the w^{l+1} weight matrix of the $(l+1)$ th layer. This sounds complicated, but let me break it down. Suppose we know the error δ^{l+1} of the $(l+1)$ layer. When we use the transpose weight matrix $(w^{l+1})^T$, we can think of it as carrying the error back across the network by giving some error metric at the output of layer l . Then we put the point object O for the point object.

This changes the error correction from the activation function in layer l and gives us an error δ^l in the input weight for layer l . Combining (eq 1) and (eq 2), we can calculate the error δ^l of each layer in the network. We first use δ^{L-1} , then (eq 2) we recalculate δ^{L-2} , and thus back through the network.

3. The rate of change of error with deviation in the network (eq 3)

$$\frac{\partial E}{\partial b_j^l} = \delta_j^l$$

Fig 2.2e

That is, the error δ_l is exactly equal to the rate of change $\partial E / \partial b_l$. The (eq 1) and (eq 2) already give us δ_l . We can simplify (eq 3) as:

$$\frac{\partial E}{\partial w_{jk}^l} = a_k^{l-1} \delta_j$$

Fig 2.2f

Where it is understood δ is being evaluated at the same neuron as the bias b .

4. Rate of change of the error with respect to any weight in the network (eq 4)

$$\frac{\partial E}{\partial b} = \delta$$

Fig 2.2g

This shows how to calculate the partial derivative $\partial E / \partial w_{ljk}$ from the quantities δ_l and a_{l-1} , which we already know how to calculate. Here a_{l-1} is the activation value of the input neuron for the weight w and δ_l is the error of the input neuron for the weight w . From the analysis (eq 4), we can say that when $a_{l-1} \approx 0$, the gradient time will also be small, which means that the learning weight is slow or the low rate does not change much. In other words, we can say that the result (eq 4) is the weighting of the neurons with the lowest learning delay. As a result, you now see that if the input neuron is not working or the electronics are satisfied, for example, the weight will learn slowly.

It seems that these four basic equations apply to all activations, not just the sigmoid or perceptron model we discussed at the beginning. Let's write this in pseudo-algorithm:

Input x : Set connection a_1 to input operation.

Forecast: calculate $e_l = w_l a_{l-1} + b_l$ and $a_l = \sigma(e_l)$ for each $l = 2, 3, \dots, L$.

Output error δ_L : Calculate the vector $\delta_L = \Delta a_L O \sigma'(e_L)$.

Backpropagation error: Calculate $\delta_l = ((w_{l+1})^T \delta_{l+1} + 1) O \sigma'(e_l)$ for each $l = L-1, L-2, \dots, 2$.

Output: The gradient of the error function is given by $\partial E / \partial w_{ljk} = a_{l-1} k \delta_l$ and $\partial E / \partial b_l = \delta_l$. Check out the algorithm and you'll understand why it's called backpropagation. We calculate the error vector δ_L from the last layer.

2.3 CNN

Convolutional neural networks (CNNs) can play an important role in the early detection of breast cancer by analyzing and interpreting clinical data such as mammograms or histopathology images. Here are a few ways CNNs can help with early detection:

1. Image Analysis: CNNs excel at image analysis tasks by learning and extracting relevant features from medical imaging. For breast cancer diagnosis, CNNs can analyze mammograms and identify abnormalities or patterns associated with malignancy, microcalcifications or structural deformations. By highlighting these areas, radiologists can target potential cancer cells and ensure early detection.

2. Risk Classification: CNNs can also assist in risk stratification by assessing the incidence of breast cancer. By training a CNN model on a large dataset containing various risk factors and clinical data, they were able to predict a person's risk of breast cancer. This can lead to early screening strategies and increased self-protection for high-risk individuals.

3. Integration with medical data: CNNs can be combined with other medical data such as patient, family history and genetic data to provide greater analytical control for early detection. By incorporating this information into CNN models, the accuracy and reliability of cancer diagnosis and risk assessment can be improved.

4. Decision-making systems: CNNs can be integrated into computer-aided diagnostics (CAD) systems to provide radiologists with decision-making. CNN models can automatically generate diagnostic information by analyzing medical images and clinical data, or suggest areas of interest, helping radiologists perform accurate and timely checks.

Overall, CNNs have the potential to improve early detection of breast cancer by analyzing clinical images, aiding in lesion detection, risk assessment, and providing decision support. This technology can help doctors detect cancer at an early stage, enabling timely intervention and improving patient outcomes.

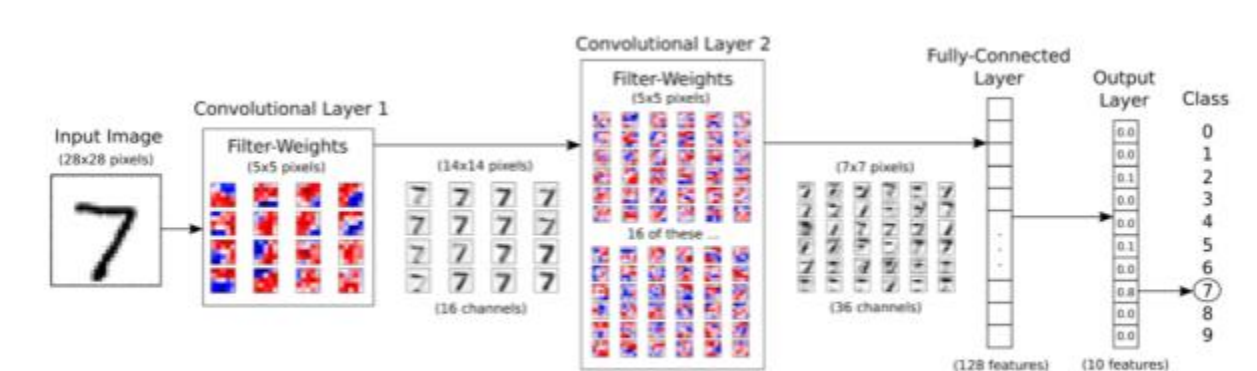


Fig 2.3a

Why CNN?

Convolutional neural networks (CNNs) are particularly suitable for tasks related to image analysis and pattern recognition. Here are a few reasons why CNNs are often used in cancer diagnosis and classification:

1. Hierarchical feature extraction: CNNs are designed to recognize and extract relevant features from a photograph. The hierarchical nature of CNNs allows them to catch local patterns and gradually learn more complex and abstract problems. In the case of a breast cancer diagnosis, CNNs can learn to distinguish from mammograms or histopathology images such as tumor shapes, microcalcifications or tissue models showing cancer centers.

2. Parameter Sharing: CNN uses the concept of parameter sharing, which reduces the number of untrained parameters. By sharing weights across the network, CNNs are more efficient and can learn from smaller training samples. In the context of breast cancer tumor classification, where registration data may be limited, CNNs can learn effectively from existing data and effectively expand on the unseen.

3. Transfer learning: CNNs can use pre-trained models on large datasets like ImageNet that are trained on millions of images. This adaptive learning allows CNNs to be launched with features already learned, increasing their performance in new tasks even with limited training data. Breast cancer classification models can take advantage of learning representations of generic images, using a pre-trained CNN model as a starting point, thereby improving their accuracy and good job performance.

CNNs can provide insight into discrimination learned during training. By visualizing trained filters or maps, researchers and doctors can better understand which pattern to focus on when diagnosing cancer. This interpretation helps build confidence in decision models and validate their validity. CNNs have many advantages for cancer diagnosis and classification, such as learning hierarchical features, resolving conflicts, coordinating for effective learning, using training modifications to improve performance, and providing reliable explanations. These features make CNN a powerful tool for early detection and accurate diagnosis of cancer, ultimately helping to improve patient outcomes.

2.4 Convolutional Layers

As mentioned earlier, folds happen in layers. Instead, the kernel, or what we call a filter here, moves to a different place in the image, changing the way the image is matched. For each function of the filter, calculate the product of the filter and the image pixel under the filter, resulting in a single pixel in the output image. Thus, moving the filter over the input image causes a new image to be created. These images are called feature maps.

Feature maps created in the first convolutional layer are down sampled. This particular map is passed to the second convolution layer. Here, filter weights must be applied for each of these newly created images. The resulting image is further down sampled. If you want to know more about how convolution works in images, you can refer to this [blog post](#) on how convolution works.

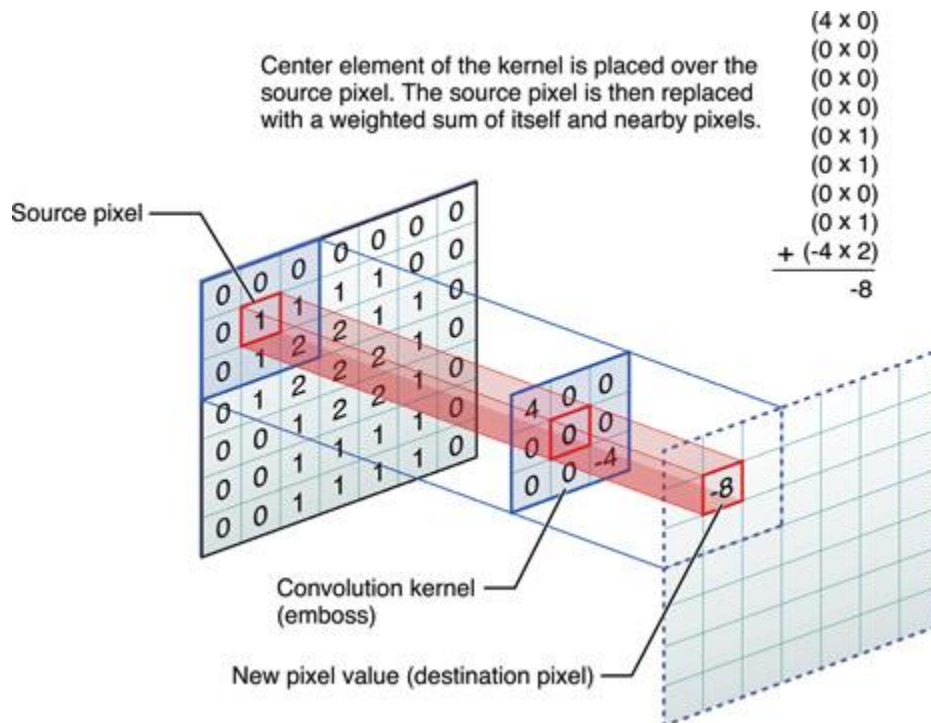


Fig 2.4a

2.5 Pooling Layer

Now, instead of down sampling by changing the convolution step, there is another reliable way to down sample an image, such as using a pooling layer. Pooling layers reduce data dimensionality by combining the output of groups of neurons in one layer with individual neurons in the next layer. Native integration combines small units, usually 2×2 . This adds half the solution. There are two types of pooling:

Max Pooling - extracts the maximum value of each group of neurons in the previous layer for each unique map

Average Pooling - extracts the average value of each group of neurons for each unique report

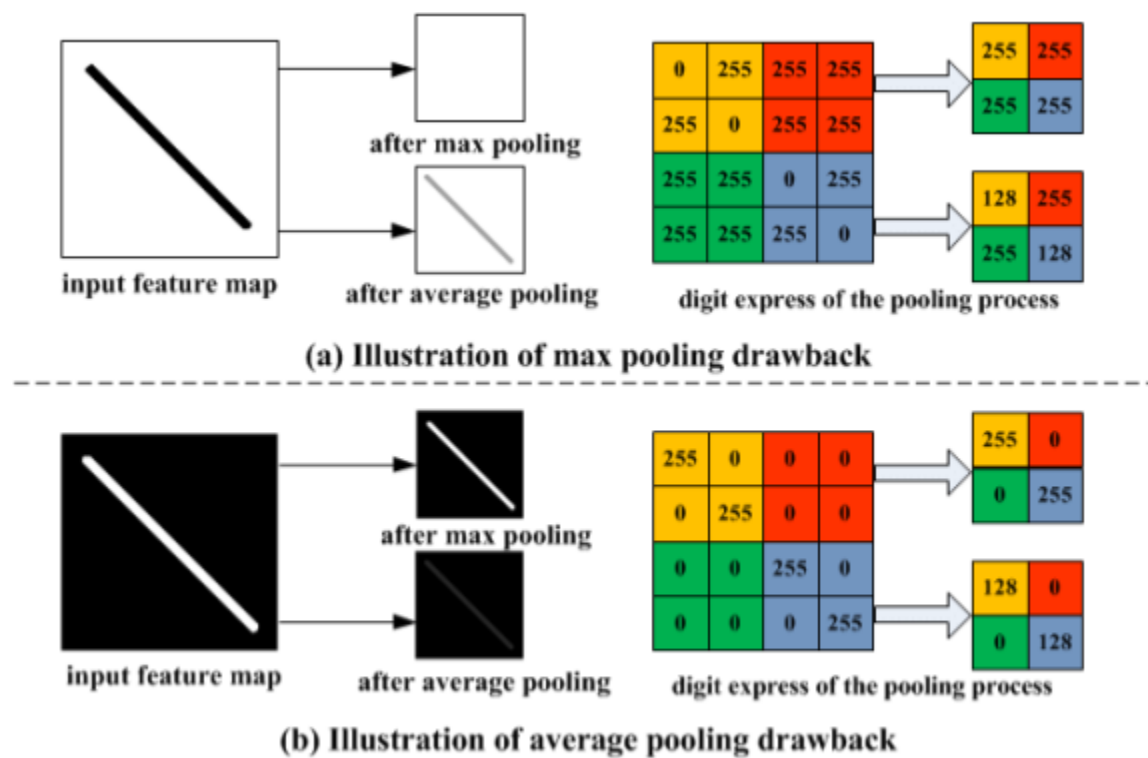


Fig 2.5a

Maximum pooling is often preferred as it minimizes noise and size.

Pooling helps remove location-independent key features. In addition, size reduction reduces the power consumption required to process data.

2.6 Fully Connected Layers

The last layer is a fully connected layer that divides our image. The output of the convolutional network is then flattened into a column vector and fed into a fully connected neural network; backpropagation is used for each iteration of the training.

Often times, the model can distinguish important features from low-level features in the image and classify them using the SoftMax classification technique.

SoftMax distribution:

The SoftMax distribution, also known as the normalized exponential function, is a numerical function that converts a vector of real numbers into a probability distribution. It is often used in machine learning and deep learning models for multitasking in the classroom.

Given an input vector of real values, the SoftMax function calculates the exponent of each item and then normalizes the resulting values so that their sum is 1. This normalization converts the values to the result where each value represents a similar value. value class.

$z = [z_1, z_2,$

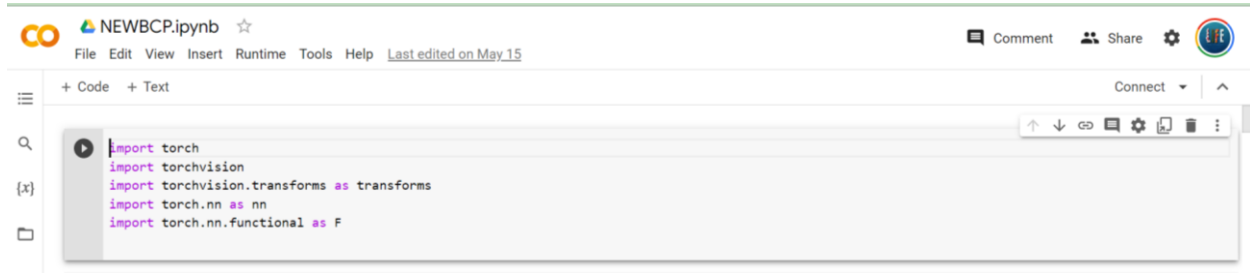
The SoftMax distribution has useful features such as variance, guaranteed negative probability, and sum up to 1.

3.Methodology:

Snippets of our Code and Outputs: -

Building our own Neural Network

Step 1: Importing the python libraries



```
import torch
import torchvision
import torchvision.transforms as transforms
import torch.nn as nn
import torch.nn.functional as F
```

Step 2: Downloading the training and testing dataset



```
[ ] transform = transforms.Compose(
    [transforms.ToTensor(),
     transforms.Normalize((0.5,), (0.5,))]
)

trainset = torchvision.datasets.MNIST(
    root='./data', train=True, download=True, transform=transform)
trainloader = torch.utils.data.DataLoader(trainset, batch_size=64,
                                          shuffle=True, num_workers=2)

[ ] testset = torchvision.datasets.MNIST(
    root='./data', train=False, download=True, transform=transform)
testloader = torch.utils.data.DataLoader(testset, batch_size=50,
                                         shuffle=False, num_workers=2)
```

Step 3: Visualizing the training images which are going to be used as input



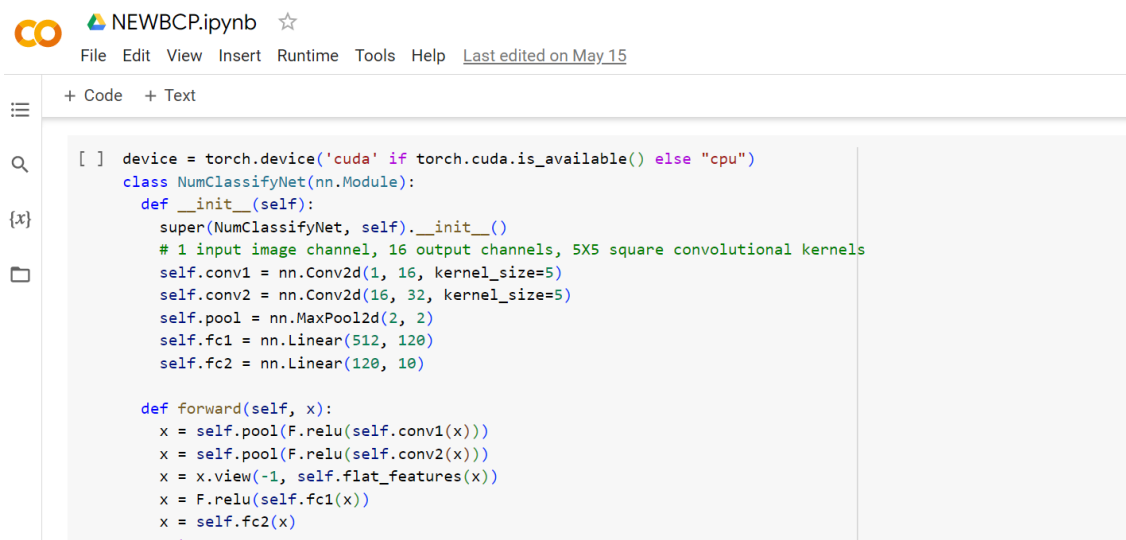
The image shows a Jupyter Notebook interface with the title 'NEWBCP.ipynb'. The code in the cell is as follows:

```
import matplotlib.pyplot as plt
import numpy as np
# functions to change tensor to numpy image
def imshow(img):
    npimg = img.numpy()
    plt.imshow(np.transpose(npimg, (1, 2, 0)))
    plt.show()

# get some random training images
dataiter = iter(trainloader)
images, labels = next(dataiter)
# show images
imshow(torchvision.utils.make_grid(images[:6], nrow=3))
```

Below the code, there is a horizontal bar with a scale from 0 to 80.

Step 4: Building the Convolution Neural Network

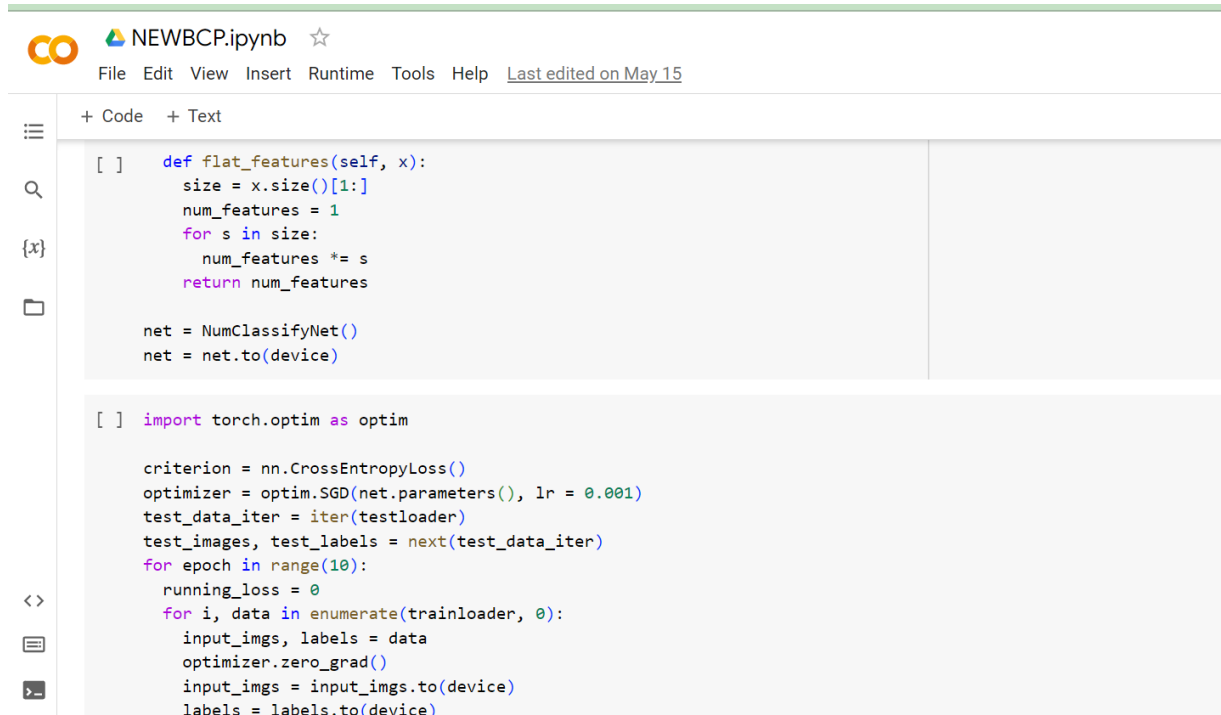


The image shows a Jupyter Notebook interface with the title 'NEWBCP.ipynb'. The code in the cell is as follows:

```
[ ] device = torch.device('cuda' if torch.cuda.is_available() else "cpu")
class NumClassifyNet(nn.Module):
    def __init__(self):
        super(NumClassifyNet, self).__init__()
        # 1 input image channel, 16 output channels, 5X5 square convolutional kernels
        self.conv1 = nn.Conv2d(1, 16, kernel_size=5)
        self.conv2 = nn.Conv2d(16, 32, kernel_size=5)
        self.pool = nn.MaxPool2d(2, 2)
        self.fc1 = nn.Linear(512, 120)
        self.fc2 = nn.Linear(120, 10)

    def forward(self, x):
        x = self.pool(F.relu(self.conv1(x)))
        x = self.pool(F.relu(self.conv2(x)))
        x = x.view(-1, self.flat_features(x))
        x = F.relu(self.fc1(x))
        x = self.fc2(x)
        return x
```

Step 5 : Training the model



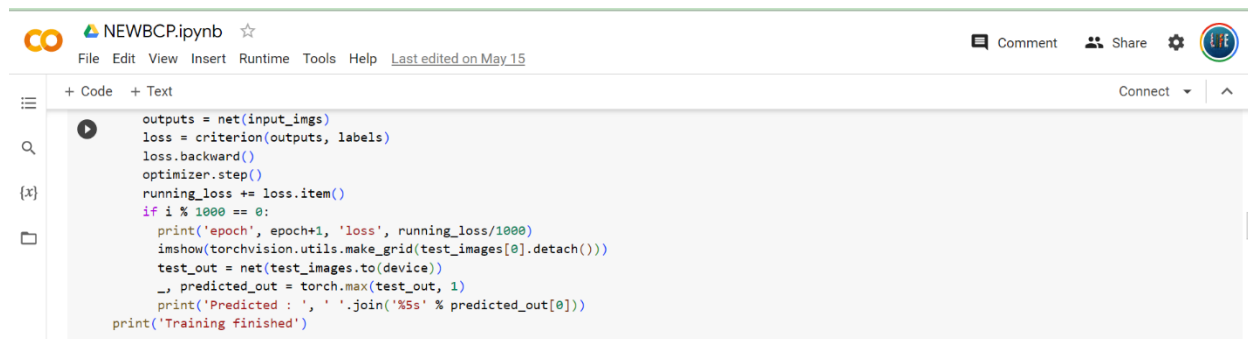
The image shows a Jupyter Notebook interface with the title "NEWBCP.ipynb". The menu bar includes "File", "Edit", "View", "Insert", "Runtime", "Tools", and "Help", with a note "Last edited on May 15". The left sidebar contains icons for file operations. The main code area has two cells. The first cell defines a function `flat_features` and initializes a `NumClassifyNet` model. The second cell imports `torch.optim` and sets up the training loop with a `range(10)` for epochs.

```
[ ] def flat_features(self, x):
    size = x.size()[1:]
    num_features = 1
    for s in size:
        num_features *= s
    return num_features

net = NumClassifyNet()
net = net.to(device)

[ ] import torch.optim as optim

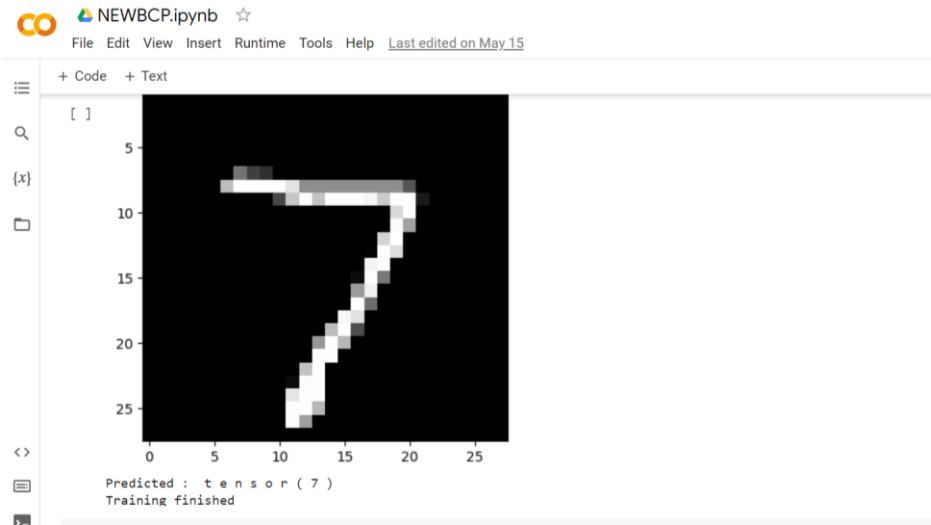
criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(net.parameters(), lr = 0.001)
test_data_iter = iter(testloader)
test_images, test_labels = next(test_data_iter)
for epoch in range(10):
    running_loss = 0
    for i, data in enumerate(trainloader, 0):
        input_imgs, labels = data
        optimizer.zero_grad()
        input_imgs = input_imgs.to(device)
        labels = labels.to(device)
```



The image shows the continuation of the Jupyter Notebook. The top bar now includes "Comment", "Share", and a "Connect" button. The code cell continues the training loop, calculating loss, updating the optimizer, and printing progress every 1000 iterations. It also includes a visualization step using `imshow` and a final prediction print statement.

```
outputs = net(input_imgs)
loss = criterion(outputs, labels)
loss.backward()
optimizer.step()
running_loss += loss.item()
if i % 1000 == 0:
    print('epoch', epoch+1, 'loss', running_loss/1000)
    imshow(torchvision.utils.make_grid(test_images[0].detach()))
    test_out = net(test_images.to(device))
    _, predicted_out = torch.max(test_out, 1)
    print('Predicted : ', ' '.join('%5s' % predicted_out[0]))
print('Training finished')
```

Runs 10 times as range is 10 , gives 10 outputs and then the training is done.



Step 6:

- Setting up Kaggle API access
- Saving the token in drive
- Mounting google drive to colab
- Configuring Kaggle environment using OS
- Downloading the dataset

NEWBCP.ipynb ☆

File Edit View Insert Runtime Tools Help Last edited on May 15

+ Code + Text

+ Code + Text

[]

```
from google.colab import drive
drive.mount('/content/gdrive')
```

Drive already mounted at /content/gdrive; to attempt to forcibly remount, call drive.mount("/content/gdrive", force_remount=True).

[]

```
import os
os.environ['KAGGLE_CONFIG_DIR'] = "/content/gdrive/MyDrive/kaggle"
```

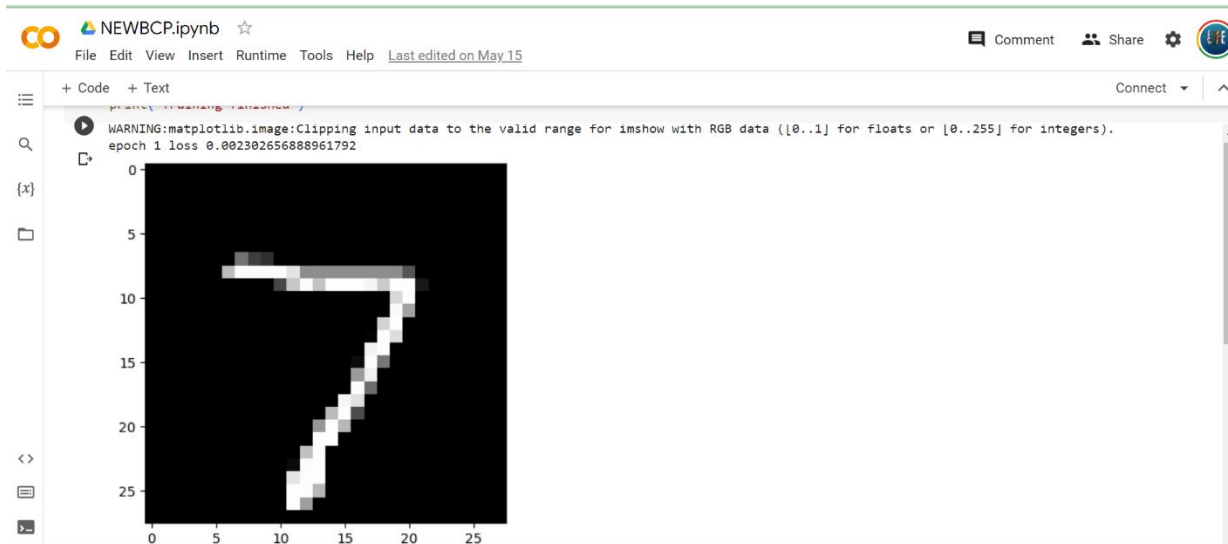
[]

```
os.chdir('../content/gdrive/MyDrive/kaggle')
!kaggle datasets download -d paultimothymooney/breast-histopathology-images
```

Downloading breast-histopathology-images.zip to /content/gdrive/MyDrive/kaggle

100% 3.09G/3.10G [00:49<00:00, 122MB/s]

100% 3.10G/3.10G [00:49<00:00, 67.5MB/s]



Step 7: Building the network now that the dataset is downloaded and converting the images to tensor

```
from torchvision.datasets import ImageFolder
import torch.nn as nn
import torch.nn.functional as F
from torch.utils.data import random_split
import torch.optim as optim
from collections import Counter
import matplotlib.pyplot as plt
import numpy as np
data_dir = "/content/gdrive/MyDrive"
folder_name = "kaggle"
image_folders = os.path.join(data_dir, folder_name)

transform = transforms.Compose([transforms.Resize((50, 50)), transforms.ToTensor()])
images = []
for file in os.listdir(image_folders):
    try:
        images.append(ImageFolder(os.path.join(image_folders, file), transform=transform))
    except:
        print(file)
datasets = torch.utils.data.ConcatDataset(images)
```

kaggle.json
breast-histopathology-images.zip

Step 8: Finding the number of samples in each class
(screenshot with output)



A Jupyter Notebook interface titled 'NEWBCP.ipynb'. The code cell contains a loop that iterates over datasets, counts the number of images for each class, and prints the results. The output cell shows the total number of images for each class: Class 0 (No Breast Cancer) with 4097 images and Class 1 (Breast Cancer present) with 737 images.

```
i=0
for dataset in datasets.datasets:
    if i==0:
        result = Counter(dataset.targets)
        i += 1
    else:
        result += Counter(dataset.targets)

result = dict(result)
print("""Total Number of Images for each Class:
Class 0 (No Breast Cancer): {}
Class 1 (Breast Cancer present): {}""".format(result[0], result[1]))
```

Total Number of Images for each Class:
Class 0 (No Breast Cancer): 4097
Class 1 (Breast Cancer present): 737

Step 9: Splitting the dataset 25% of the testing set and 75% of the training set.



A Jupyter Notebook interface titled 'NEWBCP.ipynb'. The code cell contains a loop that iterates over datasets, counts the number of images for each class, and prints the results. The output cell shows the total number of images for each class: Class 0 (No Breast Cancer) with 4097 images and Class 1 (Breast Cancer present) with 737 images. The code cell also contains a loop that iterates over datasets, counts the number of images for each class, and prints the results. The output cell shows the total number of images for each class: Class 0 (No Breast Cancer) with 4097 images and Class 1 (Breast Cancer present) with 737 images.

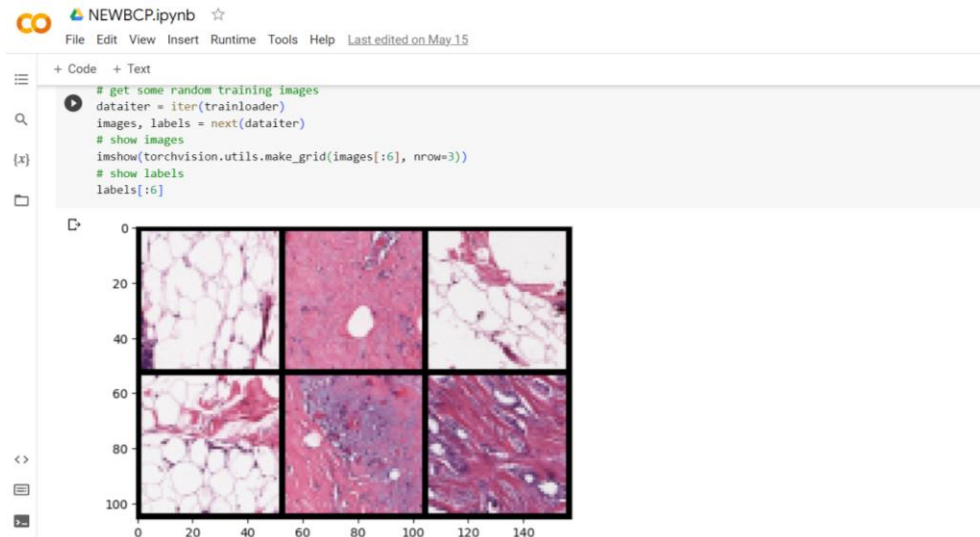
```
Class 0 (No Breast Cancer): {}
Class 1 (Breast Cancer present): {}""".format(result[0], result[1]))

Total Number of Images for each Class:
Class 0 (No Breast Cancer): 4097
Class 1 (Breast Cancer present): 737

[ ] random_seed = 42
torch.manual_seed(random_seed)

test_size = int(0.25*(result[0]+result[1]))
print(test_size)
train_size = len(datasets) - test_size
train_dataset, test_dataset = random_split(datasets, [train_size, test_size])
trainloader = torch.utils.data.DataLoader(train_dataset, batch_size=128,
                                           shuffle=True, num_workers=2)
testloader = torch.utils.data.DataLoader(test_dataset, batch_size=64,
                                          shuffle=False, num_workers=2)
```

1208



Step 10: Using the GPU and building the breast cancer neural network

The image shows a Jupyter Notebook interface for a file named 'NEWBCP.ipynb'. The code cell contains the following Python code:

```
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')

class BreastCancerClassifyNet(nn.Module):
    def __init__(self):
        super(BreastCancerClassifyNet, self).__init__()
        self.conv1 = nn.Conv2d(3, 64, kernel_size=3)
        self.conv2 = nn.Conv2d(64, 128, kernel_size=3)
        self.conv3 = nn.Conv2d(128, 256, kernel_size=3)
        self.pool = nn.MaxPool2d(2, 2)
        self.fc1 = nn.Linear(4096, 1024)
        self.fc2 = nn.Linear(1024, 512)
        self.fc3 = nn.Linear(512, 1)

    def forward(self, x):
        x = self.pool(F.relu(self.conv1(x)))
        x = self.pool(F.relu(self.conv2(x)))
        x = self.pool(F.relu(self.conv3(x)))
        x = x.view(-1, self.flat_features(x))
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        x = F.log_softmax(x)
        return x

    def flat_features(self, x):
        size = x.size()[1:]
        num_features = 1
        for s in size:
```

Step 11: Doing binary classification and using Binary Cross Entropy loss plus training the model

```
NEWBCP.ipynb
File Edit View Insert Runtime Tools Help Last edited on May 15

+ Code + Text

criterion = nn.BCELoss()
optimizer = optim.SGD(net.parameters(), lr = 0.001)
test_data_iter = iter(testloader)
test_images, test_labels = next(dataiter)
for epoch in range(10):
    running_loss = 0
    for i, data in enumerate(trainloader, 0):
        input_imgs, labels = data
        input_imgs = input_imgs.to(device)
        labels = labels.to(device)
        optimizer.zero_grad()
        outputs = net(input_imgs)
        labels = labels.unsqueeze(1).float()
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()

    #printing stats and checking prediction as we train
    running_loss += loss.item()
    if i % 10000 == 0:
        print('epoch', epoch+1, 'loss', running_loss/10000)
        imshow(torchvision.utils.make_grid(test_images[0].detach()))
        test_out = net(test_images.to(device))
        _, predicted_out = torch.max(test_out, 1)
        print('Predicted : ', ' '.join('%5s' % predicted_out[0]))
    print('Training finished')
```

Step 12: Calculating the accuracy and testing the trained model on dataset

```
NEWBCP.ipynb
File Edit View Insert Runtime Tools Help Last edited on May 15
Comment Share

+ Code + Text
Connect

[ ]
Predicted : tensor(0)
Training finished

correct = 0
total = 0
with torch.no_grad():
    for data in testloader:
        test_images, test_labels = data
        test_out = net(test_images.to(device))
        _, predicted = torch.max(test_out.data, 1)
        total += test_labels.size(0)
        for _id, out_pred in enumerate(predicted):
            if int(out_pred) == int(test_labels[_id]):
                correct += 1

print('Accuracy of the network on the 44252 test images: %d %%' % (
    100 * correct / total))

<ipython-input-8-93b67564b36b>:21: UserWarning: Implicit dimension choice for log_softmax has been deprecated. Change the call to include dim=X as an argument.
x = F.log_softmax(x)
Accuracy of the network on the 44252 test images: 84 %
```

Result : 84 % accuracy

Conclusion:

In rundown, this venture centers on the utilize of convolutional neural systems (CNNs) for breast cancer classification utilizing histopathology pictures. The extend points to help within the early location and determination of breast cancer by utilizing the control of profound learning and picture examination technologies. CNN show actualized utilizing Python and PyTorch system, prepared in Google Colab and empowered by CUDA to speed up computation. The Kaggle dataset contains a expansive database of histopathology pictures for preparing, approval and testing purposes. The CNN demonstrate appears great comes about, accomplishing 84% precision on the test data. This high exactness illustrates the potential of CNNs to assist specialists distinguish and classify breast cancer cells. The model's capacity to memorize and extricate critical highlights from histopathology pictures is basic to its success. In expansion, this extend advances versatile learning by leveraging information learned from huge information, employing a pre-trained ResNet50 demonstrate as a custom extractor. This versatile learning moves forward demonstrate execution and diminishes the require for preparing on expansive datasets. The comes about of this think about have critical suggestions for breast cancer diagnosis. CNN models can offer assistance radiologists and specialists distinguish cancer cells in restorative pictures, make early discovery and convenient treatment. Through the utilize of demonstrative strategies, CNNs have the potential to make strides the effectiveness, exactness and consistency of breast cancer diagnosis. However, it is critical to keep in mind that the CNN demonstrate ought to be seen as an help, not a substitute for the specialist. Last choices with respect to determination and treatment ought to continuously be made by an experienced doctor, counting other restorative and proficient information. In rundown, this extend illustrates the adequacy of CNNs in breast cancer classification utilizing histopathology pictures.

Future Work:-

Future ponders utilizing CNNs in breast cancer classification will look for to grow the database to incorporate more different and bigger datasets, fine-tune and optimize hyperparameters to progress execution, coordinated clinical information for investigation and chance evaluation, look Multimodal information combination moves forward exactness, progresses precision . encouraging collaboration with doctors to interpret and translate decision-making models, approve models in real-world clinical settings through thorough testing, and take after up as therapeutic needs and administrative prerequisites. These progresses offer assistance make strides symptomatic exactness, early location, and personalized treatment techniques for cancer patients.

References :-

1. Esteva, A., Kuprel, B., Novoa, R. A., Ko, J., Swetter, S. M., Blau, H. M., & Thrun, S
2. Cruz-Roa, A., Basavanthally, A., González, F. A., Gilmore, H., Feldman, M., Ganesan, S., ... & Madabhushi, A. (2013). Automatic detection of invasive ductal carcinoma in whole slide images with convolutional neural networks. In *Medical Imaging 2013: Digital Pathology* (Vol. 8676, p. 86760X). International Society for Optics and Photonics.
3. Ren, J., Chung, A. C., Xiong, Z., & Yang, J. (2020). Breast cancer detection from mammograms using deep convolutional neural networks. *International Journal of Machine Learning and Cybernetics*, 11(8), 1991-2003.
4. Wang, W., Li, J., Zhang, J., Sun, W., Chen, Y., & Zhang, Y. (2019). Breast cancer histopathological image classification using convolutional neural networks with small SE-inception modules. *IEEE Access*, 7, 101980-101992.
5. Wang, H., Xu, Y., Jiang, Y., Zhang, L., Chen, Y., & Zhang, Y. (2021). Automated breast cancer diagnosis on histopathological images using an improved convolutional neural network. *Frontiers in Oncology*, 11, 603571.
6. Wang, W., Li, X., & Yao, J. (2020). Breast cancer detection based on deep learning models: A survey. *Medical Image Analysis*, 60, 101594.
7. Yu, Y., Xu, X., Liu, F., Zhang, Y., & Tang, J. (2021). A hybrid model based on convolutional neural network and fuzzy support vector machine for breast cancer diagnosis. *Expert Systems with Applications*, 174, 114731.
8. Wei, J. W., Zhang, T., Lv, W. B., & Ma, Y. T. (2020). Diagnosis of breast cancer using a hybrid model combining convolutional neural network and principal component analysis. *Biomedical Signal Processing and Control*, 58, 101837.
9. Li, X., Li, W., Zhang, W., Li, J., Liu, Z., & Yao, J. (2020). Breast cancer classification using deep convolutional neural networks and multimodal imaging. *IEEE Journal of Biomedical and Health Informatics*, 24(6), 1826-1833.
10. Park, H. J., Kim, J. H., Kim, H. J., & Lee, J. Y. (2019). Breast cancer detection using deep convolutional neural networks and support vector machines. *Journal of Medical Systems*, 43(10), 304.