Enhancement 5: Choose a new dataset from the list below. Search the Internet and download your chosen dataset (many of them could be available on kaggle). Adapt your model to your dataset. Train your model and record your results.

- cancer_dataset - Breast cancer dataset.
- crab_dataset - Crab gender dataset.
- glass_dataset - Glass chemical dataset.
- iris_dataset - Iris flower dataset.
- ovarian_dataset - Ovarian cancer dataset.
- thyroid_dataset - Thyroid function dataset.

Imports

In [47]:
```
import pandas as pd
import torch
import torch.nn as nn
import torch.nn.functional as F
from torch.optim import AdamW
from torch.utils.data import Dataset, DataLoader
from tqdm.notebook import tqdm
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
```

Load Dataset

In [48]:
```
from sklearn.datasets import load_breast_cancer
cancer_dataset = load_breast_cancer()
```

In [49]:
```
type(cancer_dataset)
```

Out[49]:
```
sklearn.utils.Bunch
```

In [50]:
```
df = pd.DataFrame(data=cancer_dataset.data, columns=cancer_dataset.feature_names)
df['target'] = cancer_dataset.target
```

In [51]:
```
# make sure it looks good
df.head()
# df.tail()
```

Out[51]:

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	mean symmetry	f dime
0	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.14710	0.2419	0.
1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.07017	0.1812	0.
2	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.12790	0.2069	0.
3	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.10520	0.2597	0.
4	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.10430	0.1809	0.

5 rows × 31 columns

In [52]:
```
# how many features?
# WOW!
len(df.columns) -1
```

Out[52]:
```
30
```

In [53]:
```
# 0 means malignant
# 1 means benign
# @$@ switch these
df['target'].tail(5)
```

Out[53]:
```
564    0
565    0
566    0
567    0
568    1
Name: target, dtype: int32
```

In [54]:
```
# normalize data so that each feature has a mean of 0 and a std of 1
labels = df['target']
# @$@ there is a better way
# if vals too big. -> maybe each col normailzed seperatly
df = (df - df.mean()) / df.std()
df['target'] = labels
```

In [55]:
```
df.head()
```

Out[55]:

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	me symmet
0	1.096100	-2.071512	1.268817	0.983510	1.567087	3.280628	2.650542	2.530249	2.2155
1	1.828212	-0.353322	1.684473	1.907030	-0.826235	-0.486643	-0.023825	0.547662	0.0013
2	1.578499	0.455786	1.565126	1.557513	0.941382	1.052000	1.362280	2.035440	0.9388
3	-0.768233	0.253509	-0.592166	-0.763792	3.280667	3.399917	1.914213	1.450431	2.8648
4	1.748758	-1.150804	1.775011	1.824624	0.280125	0.538866	1.369806	1.427237	-0.0095

5 rows × 31 columns

In [56]:

```python
# sumamry statistics of the data
df.describe()
```

Out[56]:

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	
count	5.690000e+02	5.690000e+02	5.690000e+02	5.690000e+02	5.690000e+02	5.690000e+02	5.6
mean	-3.142575e-15	-6.558316e-15	-7.012551e-16	-8.339355e-16	6.083788e-15	-1.081346e-15	-3.
std	1.000000e+00	1.000000e+00	1.000000e+00	1.000000e+00	1.000000e+00	1.000000e+00	1.0
min	-2.027864e+00	-2.227289e+00	-1.982759e+00	-1.453164e+00	-3.109349e+00	-1.608721e+00	-1.1
25%	-6.887793e-01	-7.253249e-01	-6.913472e-01	-6.666089e-01	-7.103378e-01	-7.464292e-01	-7.
50%	-2.148925e-01	-1.045442e-01	-2.357726e-01	-2.949274e-01	-3.486040e-02	-2.217454e-01	-3.
75%	4.689800e-01	5.836621e-01	4.992377e-01	3.631877e-01	6.356397e-01	4.934227e-01	5.
max	3.967796e+00	4.647799e+00	3.972634e+00	5.245913e+00	4.766717e+00	4.564409e+00	4.2

8 rows × 31 columns

Load this dataset for training a neural network

In [57]:

```python
# The dataset class
# inherit the Dataset class and create a new class CancerDataset
class CancerDataset(Dataset):
    def __init__(self, df):
        self.df = df
        self.features = []
        self.labels = []

        # Iterate through rows of the DataFrame
        for _, row in df.iterrows():
            # Append features for each row except the 'target' column
            self.features.append(row.drop('target').tolist())
            # Append label for each row
            self.labels.append(row['target'])
    # set length to reaturn number of rows
    def __len__(self):
        return len(self.df)

    # get a sample in the form of a dictionary (features and labels) given its index
    def __getitem__(self, idx):
        if torch.is_tensor(idx):
            idx = idx.tolist()

        features = self.features[idx]
        features = torch.FloatTensor(features)

        labels = torch.tensor(int(self.labels[idx]), dtype = torch.long)

        return {'labels': labels, 'features': features}

# instanciate a CancerDataset object based off of data_df
cancer_dataset = CancerDataset(df)
# perform a 80, 10, 10 split using the cancer_dataset object
train_dataset, val_dataset, test_dataset = torch.utils.data.random_split(cancer_datase

# The dataloader
train_dataloader = DataLoader(
    # dataset input
    train_dataset,
    # batches of 4 during training
    batch_size = 4,
    # don't learn from the order of the dataset!
    shuffle = True,
    # multy processing. (0 = 1 process to load data)
    # how many processes can you do?
    num_workers = 0
)

#want results without any randomness, therfore shuffle is False for these
val_dataloader = DataLoader(val_dataset, batch_size = 4, shuffle = False, num_workers
test_dataloader = DataLoader(test_dataset, batch_size = 4, shuffle = False, num_worker
```

In [58]:
```
# peak into the dataset
for i in cancer_dataset:
    print(i)
    break
```

```
{'labels': tensor(0), 'features': tensor([ 1.0961, -2.0715,  1.2688,  0.9835,  1.567
1,  3.2806,  2.6505,  2.5302,
         2.2156,  2.2538,  2.4875, -0.5648,  2.8305,  2.4854, -0.2138,  1.3157,
         0.7234,  0.6602,  1.1477,  0.9063,  1.8850, -1.3581,  2.3016,  1.9995,
         1.3065,  2.6144,  2.1077,  2.2941,  2.7482,  1.9353])}
```

Neural Network

In [59]:
```
# change the device to gpu if available
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
```

In [60]:
```
#defines new class named CancerModel, inherits from torch.blah
class CancerModel(torch.nn.Module):
    def __init__(self):
        # initialize in the same way the parent class does first
        super(CancerModel, self).__init__()

        # create linear layer. 11 inputs 200 outputs?
        self.linear1 = torch.nn.Linear(30, 210)
        # creates activation function (What does this do and how does it work @$@)
        self.activation = torch.nn.ReLU()

        # a second linear layer. Takes in 200 and outputs 6 (one corresponding to each
        self.linear2 = torch.nn.Linear(210, 2)

        # sigmoid activation for binary
        self.softmax = torch.nn.Softmax(dim=1)
    # defines forward pass of nn
    # sends input through the tunnel: lin1 -> act -> line2 -> softmax
    def forward(self, x):
        x = self.linear1(x)
        x = self.activation(x)
        x = self.linear2(x)
        x = self.activation(x)
        x = self.softmax(x)
        return x

# earlier we set device to cpu or gpu
# create this CancerModel object, then move to cpu or gpu
cancermodel = CancerModel().to(device)
```

In [61]:

```
# Define and the loss function and optimizer
# we are trying binary cross entropy loss
# what is BCE using as inputs
    #
criterion = nn.CrossEntropyLoss().to(device)

# an optimizer @$@ what does it do?
# specify the learning rate
optimizer = AdamW(cancermodel.parameters(), lr = 1e-3)
```

In [62]:

```
# Lets define the training steps
def accuracy(preds, labels):
    preds = torch.argmax(preds, dim=1).flatten()
    return torch.sum(preds == labels) / len(labels)

def train(model, data_loader, optimizer, criterion):
    epoch_loss = 0
    epoch_acc = 0

    # activate training mode
    model.train()

    # iterate over batches
    for d in tqdm(data_loader):
        inputs = d['features'].to(device)
        #labels = d['labels'].unsqueeze(1).to(device)  # Adjusting the shape of the ta
        labels = d['labels'].to(device)  # Adjusting the shape of the target tensor

        #labels = labels.float()  # Cast labels to float32

        # obtaining predictions
        outputs = cancermodel(inputs)
        preds = (outputs > 0.5).float()  # Convert logits to binary predictions
        # calulate loss and accuracy by comparing predictions to labels
        loss = criterion(outputs, labels)
        acc = accuracy(preds, labels)  # Use binary accuracy function

        # Backpropagation
        loss.backward()
        optimizer.step()
        optimizer.zero_grad()

        epoch_loss += loss.item()
        epoch_acc += acc.item()

    return epoch_loss / len(data_loader), epoch_acc / len(data_loader)


# Lets define the testing steps
def evaluate(model, data_loader, criterion):
    epoch_loss = 0
    epoch_acc = 0
    epoch_precision = 0
    epoch_recall = 0
    epoch_f1 = 0

    # eval mode activate!
    model.eval()

    # iterate over batches (later we will average the huge sum)
    with torch.no_grad():
        for d in data_loader:
            # extract labels and features from batch and send to cpu/gpu
            inputs = d['features'].to(device)
            labels = d['labels'].to(device)  # Adjusting the shape of the target tens
            # labels = labels.float()  # Convert labels to float32

            # use model to get predictions
            outputs = cancermodel(inputs)

            #  get loss and accuracy
            loss = criterion(outputs, labels)
            acc = accuracy(outputs, labels)

            # run a big sum!
            epoch_loss += loss.item()
            epoch_acc += acc.item()
            precision = precision_score(labels.cpu(), outputs.argmax(dim=1).cpu() > 0.
            recall = recall_score(labels.cpu(), outputs.argmax(dim=1).cpu() > 0.5, ave
            f1 = f1_score(labels.cpu(), outputs.argmax(dim=1).cpu() > 0.5, average='we
            epoch_precision += precision
            epoch_recall += recall
            epoch_f1 += f1

    # average big sums by # of epochs
    num_batches = len(data_loader)
    return (epoch_loss / num_batches,
            epoch_acc / num_batches,
            epoch_precision / num_batches,
            epoch_recall / num_batches,
            epoch_f1 / num_batches)
```

```
In [63]: # Let's train our model
         for epoch in range(50):
             train_loss, train_acc = train(cancermodel, train_dataloader, optimizer, criterion)
             valid_loss, valid_acc, valid_precision, valid_recall, valid_f1 = evaluate(cancermc

             print(f'| Epoch: {epoch+1:02} | Train Loss: {train_loss:.3f} | Train Acc: {train_a
```

```
  0%|          | 0/114 [00:00<?, ?it/s]
| Epoch: 01 | Train Loss: 0.440 | Train Acc: 94.74% | Val. Loss: 0.370 | Val. Acc: 9
6.67% | Val. Precision: 0.978 | Val. Recall: 0.967 | Val. F1-score: 0.964 |
  0%|          | 0/114 [00:00<?, ?it/s]
| Epoch: 02 | Train Loss: 0.351 | Train Acc: 98.03% | Val. Loss: 0.355 | Val. Acc: 9
6.67% | Val. Precision: 0.978 | Val. Recall: 0.967 | Val. F1-score: 0.964 |
  0%|          | 0/114 [00:00<?, ?it/s]
| Epoch: 03 | Train Loss: 0.339 | Train Acc: 98.68% | Val. Loss: 0.351 | Val. Acc: 9
6.67% | Val. Precision: 0.978 | Val. Recall: 0.967 | Val. F1-score: 0.964 |
  0%|          | 0/114 [00:00<?, ?it/s]
| Epoch: 04 | Train Loss: 0.332 | Train Acc: 99.12% | Val. Loss: 0.348 | Val. Acc: 9
6.67% | Val. Precision: 0.978 | Val. Recall: 0.967 | Val. F1-score: 0.964 |
  0%|          | 0/114 [00:00<?, ?it/s]
| Epoch: 05 | Train Loss: 0.329 | Train Acc: 99.12% | Val. Loss: 0.346 | Val. Acc: 9
6.67% | Val. Precision: 0.978 | Val. Recall: 0.967 | Val. F1-score: 0.964 |
  0%|          | 0/114 [00:00<?, ?it/s]
| Epoch: 06 | Train Loss: 0.328 | Train Acc: 99.12% | Val. Loss: 0.346 | Val. Acc: 9
6.67% | Val. Precision: 0.978 | Val. Recall: 0.967 | Val. F1-score: 0.964 |
  0%|          | 0/114 [00:00<?, ?it/s]
| Epoch: 07 | Train Loss: 0.326 | Train Acc: 99.12% | Val. Loss: 0.349 | Val. Acc: 9
6.67% | Val. Precision: 0.978 | Val. Recall: 0.967 | Val. F1-score: 0.964 |
  0%|          | 0/114 [00:00<?, ?it/s]
| Epoch: 08 | Train Loss: 0.325 | Train Acc: 99.34% | Val. Loss: 0.349 | Val. Acc: 9
6.67% | Val. Precision: 0.978 | Val. Recall: 0.967 | Val. F1-score: 0.964 |
  0%|          | 0/114 [00:00<?, ?it/s]
| Epoch: 09 | Train Loss: 0.324 | Train Acc: 99.34% | Val. Loss: 0.348 | Val. Acc: 9
6.67% | Val. Precision: 0.978 | Val. Recall: 0.967 | Val. F1-score: 0.964 |
  0%|          | 0/114 [00:00<?, ?it/s]
| Epoch: 10 | Train Loss: 0.324 | Train Acc: 99.34% | Val. Loss: 0.348 | Val. Acc: 9
6.67% | Val. Precision: 0.978 | Val. Recall: 0.967 | Val. F1-score: 0.964 |
  0%|          | 0/114 [00:00<?, ?it/s]
| Epoch: 11 | Train Loss: 0.323 | Train Acc: 99.34% | Val. Loss: 0.346 | Val. Acc: 9
6.67% | Val. Precision: 0.978 | Val. Recall: 0.967 | Val. F1-score: 0.964 |
  0%|          | 0/114 [00:00<?, ?it/s]
| Epoch: 12 | Train Loss: 0.322 | Train Acc: 99.34% | Val. Loss: 0.347 | Val. Acc: 9
6.67% | Val. Precision: 0.978 | Val. Recall: 0.967 | Val. F1-score: 0.964 |
  0%|          | 0/114 [00:00<?, ?it/s]
| Epoch: 13 | Train Loss: 0.322 | Train Acc: 99.34% | Val. Loss: 0.347 | Val. Acc: 9
6.67% | Val. Precision: 0.978 | Val. Recall: 0.967 | Val. F1-score: 0.964 |
  0%|          | 0/114 [00:00<?, ?it/s]
| Epoch: 14 | Train Loss: 0.322 | Train Acc: 99.34% | Val. Loss: 0.347 | Val. Acc: 9
6.67% | Val. Precision: 0.978 | Val. Recall: 0.967 | Val. F1-score: 0.964 |
  0%|          | 0/114 [00:00<?, ?it/s]
| Epoch: 15 | Train Loss: 0.321 | Train Acc: 99.34% | Val. Loss: 0.348 | Val. Acc: 9
6.67% | Val. Precision: 0.978 | Val. Recall: 0.967 | Val. F1-score: 0.964 |
  0%|          | 0/114 [00:00<?, ?it/s]
| Epoch: 16 | Train Loss: 0.321 | Train Acc: 99.34% | Val. Loss: 0.347 | Val. Acc: 9
6.67% | Val. Precision: 0.978 | Val. Recall: 0.967 | Val. F1-score: 0.964 |
  0%|          | 0/114 [00:00<?, ?it/s]
| Epoch: 17 | Train Loss: 0.321 | Train Acc: 99.34% | Val. Loss: 0.347 | Val. Acc: 9
6.67% | Val. Precision: 0.978 | Val. Recall: 0.967 | Val. F1-score: 0.964 |
  0%|          | 0/114 [00:00<?, ?it/s]
```

```
| Epoch: 18 | Train Loss: 0.321 | Train Acc: 99.34% | Val. Loss: 0.345 | Val. Acc: 9
6.67% | Val. Precision: 0.978 | Val. Recall: 0.967 | Val. F1-score: 0.964 |
  0%|          | 0/114 [00:00<?, ?it/s]
| Epoch: 19 | Train Loss: 0.321 | Train Acc: 99.34% | Val. Loss: 0.347 | Val. Acc: 9
6.67% | Val. Precision: 0.978 | Val. Recall: 0.967 | Val. F1-score: 0.964 |
  0%|          | 0/114 [00:00<?, ?it/s]
| Epoch: 20 | Train Loss: 0.321 | Train Acc: 99.34% | Val. Loss: 0.347 | Val. Acc: 9
6.67% | Val. Precision: 0.978 | Val. Recall: 0.967 | Val. F1-score: 0.964 |
  0%|          | 0/114 [00:00<?, ?it/s]
| Epoch: 21 | Train Loss: 0.321 | Train Acc: 99.34% | Val. Loss: 0.347 | Val. Acc: 9
6.67% | Val. Precision: 0.978 | Val. Recall: 0.967 | Val. F1-score: 0.964 |
  0%|          | 0/114 [00:00<?, ?it/s]
| Epoch: 22 | Train Loss: 0.321 | Train Acc: 99.34% | Val. Loss: 0.347 | Val. Acc: 9
6.67% | Val. Precision: 0.978 | Val. Recall: 0.967 | Val. F1-score: 0.964 |
  0%|          | 0/114 [00:00<?, ?it/s]
| Epoch: 23 | Train Loss: 0.320 | Train Acc: 99.34% | Val. Loss: 0.347 | Val. Acc: 9
6.67% | Val. Precision: 0.978 | Val. Recall: 0.967 | Val. F1-score: 0.964 |
  0%|          | 0/114 [00:00<?, ?it/s]
| Epoch: 24 | Train Loss: 0.320 | Train Acc: 99.34% | Val. Loss: 0.347 | Val. Acc: 9
6.67% | Val. Precision: 0.978 | Val. Recall: 0.967 | Val. F1-score: 0.964 |
  0%|          | 0/114 [00:00<?, ?it/s]
| Epoch: 25 | Train Loss: 0.320 | Train Acc: 99.34% | Val. Loss: 0.347 | Val. Acc: 9
6.67% | Val. Precision: 0.978 | Val. Recall: 0.967 | Val. F1-score: 0.964 |
  0%|          | 0/114 [00:00<?, ?it/s]
| Epoch: 26 | Train Loss: 0.321 | Train Acc: 99.34% | Val. Loss: 0.348 | Val. Acc: 9
6.67% | Val. Precision: 0.978 | Val. Recall: 0.967 | Val. F1-score: 0.964 |
  0%|          | 0/114 [00:00<?, ?it/s]
| Epoch: 27 | Train Loss: 0.320 | Train Acc: 99.34% | Val. Loss: 0.346 | Val. Acc: 9
6.67% | Val. Precision: 0.978 | Val. Recall: 0.967 | Val. F1-score: 0.964 |
  0%|          | 0/114 [00:00<?, ?it/s]
| Epoch: 28 | Train Loss: 0.319 | Train Acc: 99.56% | Val. Loss: 0.346 | Val. Acc: 9
6.67% | Val. Precision: 0.978 | Val. Recall: 0.967 | Val. F1-score: 0.964 |
  0%|          | 0/114 [00:00<?, ?it/s]
| Epoch: 29 | Train Loss: 0.318 | Train Acc: 99.56% | Val. Loss: 0.345 | Val. Acc: 9
6.67% | Val. Precision: 0.978 | Val. Recall: 0.967 | Val. F1-score: 0.964 |
  0%|          | 0/114 [00:00<?, ?it/s]
| Epoch: 30 | Train Loss: 0.318 | Train Acc: 99.56% | Val. Loss: 0.346 | Val. Acc: 9
6.67% | Val. Precision: 0.978 | Val. Recall: 0.967 | Val. F1-score: 0.964 |
  0%|          | 0/114 [00:00<?, ?it/s]
| Epoch: 31 | Train Loss: 0.318 | Train Acc: 99.56% | Val. Loss: 0.346 | Val. Acc: 9
6.67% | Val. Precision: 0.978 | Val. Recall: 0.967 | Val. F1-score: 0.964 |
  0%|          | 0/114 [00:00<?, ?it/s]
| Epoch: 32 | Train Loss: 0.318 | Train Acc: 99.56% | Val. Loss: 0.346 | Val. Acc: 9
6.67% | Val. Precision: 0.978 | Val. Recall: 0.967 | Val. F1-score: 0.964 |
  0%|          | 0/114 [00:00<?, ?it/s]
| Epoch: 33 | Train Loss: 0.318 | Train Acc: 99.56% | Val. Loss: 0.347 | Val. Acc: 9
6.67% | Val. Precision: 0.978 | Val. Recall: 0.967 | Val. F1-score: 0.964 |
  0%|          | 0/114 [00:00<?, ?it/s]
| Epoch: 34 | Train Loss: 0.318 | Train Acc: 99.56% | Val. Loss: 0.346 | Val. Acc: 9
6.67% | Val. Precision: 0.978 | Val. Recall: 0.967 | Val. F1-score: 0.964 |
  0%|          | 0/114 [00:00<?, ?it/s]
| Epoch: 35 | Train Loss: 0.318 | Train Acc: 99.56% | Val. Loss: 0.346 | Val. Acc: 9
6.67% | Val. Precision: 0.978 | Val. Recall: 0.967 | Val. F1-score: 0.964 |
  0%|          | 0/114 [00:00<?, ?it/s]
| Epoch: 36 | Train Loss: 0.318 | Train Acc: 99.56% | Val. Loss: 0.347 | Val. Acc: 9
6.67% | Val. Precision: 0.978 | Val. Recall: 0.967 | Val. F1-score: 0.964 |
  0%|          | 0/114 [00:00<?, ?it/s]
| Epoch: 37 | Train Loss: 0.318 | Train Acc: 99.56% | Val. Loss: 0.347 | Val. Acc: 9
6.67% | Val. Precision: 0.978 | Val. Recall: 0.967 | Val. F1-score: 0.964 |
  0%|          | 0/114 [00:00<?, ?it/s]
```

```
| Epoch: 38 | Train Loss: 0.318 | Train Acc: 99.56% | Val. Loss: 0.347 | Val. Acc: 9
6.67% | Val. Precision: 0.978 | Val. Recall: 0.967 | Val. F1-score: 0.964 |
  0%|          | 0/114 [00:00<?, ?it/s]
| Epoch: 39 | Train Loss: 0.318 | Train Acc: 99.56% | Val. Loss: 0.347 | Val. Acc: 9
6.67% | Val. Precision: 0.978 | Val. Recall: 0.967 | Val. F1-score: 0.964 |
  0%|          | 0/114 [00:00<?, ?it/s]
| Epoch: 40 | Train Loss: 0.318 | Train Acc: 99.56% | Val. Loss: 0.347 | Val. Acc: 9
6.67% | Val. Precision: 0.978 | Val. Recall: 0.967 | Val. F1-score: 0.964 |
  0%|          | 0/114 [00:00<?, ?it/s]
| Epoch: 41 | Train Loss: 0.318 | Train Acc: 99.56% | Val. Loss: 0.346 | Val. Acc: 9
6.67% | Val. Precision: 0.978 | Val. Recall: 0.967 | Val. F1-score: 0.964 |
  0%|          | 0/114 [00:00<?, ?it/s]
| Epoch: 42 | Train Loss: 0.318 | Train Acc: 99.56% | Val. Loss: 0.347 | Val. Acc: 9
6.67% | Val. Precision: 0.978 | Val. Recall: 0.967 | Val. F1-score: 0.964 |
  0%|          | 0/114 [00:00<?, ?it/s]
| Epoch: 43 | Train Loss: 0.318 | Train Acc: 99.56% | Val. Loss: 0.347 | Val. Acc: 9
6.67% | Val. Precision: 0.978 | Val. Recall: 0.967 | Val. F1-score: 0.964 |
  0%|          | 0/114 [00:00<?, ?it/s]
| Epoch: 44 | Train Loss: 0.318 | Train Acc: 99.56% | Val. Loss: 0.347 | Val. Acc: 9
6.67% | Val. Precision: 0.978 | Val. Recall: 0.967 | Val. F1-score: 0.964 |
  0%|          | 0/114 [00:00<?, ?it/s]
| Epoch: 45 | Train Loss: 0.318 | Train Acc: 99.56% | Val. Loss: 0.347 | Val. Acc: 9
6.67% | Val. Precision: 0.978 | Val. Recall: 0.967 | Val. F1-score: 0.964 |
  0%|          | 0/114 [00:00<?, ?it/s]
| Epoch: 46 | Train Loss: 0.318 | Train Acc: 99.56% | Val. Loss: 0.347 | Val. Acc: 9
6.67% | Val. Precision: 0.978 | Val. Recall: 0.967 | Val. F1-score: 0.964 |
  0%|          | 0/114 [00:00<?, ?it/s]
| Epoch: 47 | Train Loss: 0.318 | Train Acc: 99.56% | Val. Loss: 0.347 | Val. Acc: 9
6.67% | Val. Precision: 0.978 | Val. Recall: 0.967 | Val. F1-score: 0.964 |
  0%|          | 0/114 [00:00<?, ?it/s]
| Epoch: 48 | Train Loss: 0.318 | Train Acc: 99.56% | Val. Loss: 0.348 | Val. Acc: 9
6.67% | Val. Precision: 0.978 | Val. Recall: 0.967 | Val. F1-score: 0.964 |
  0%|          | 0/114 [00:00<?, ?it/s]
| Epoch: 49 | Train Loss: 0.318 | Train Acc: 99.56% | Val. Loss: 0.347 | Val. Acc: 9
6.67% | Val. Precision: 0.978 | Val. Recall: 0.967 | Val. F1-score: 0.964 |
  0%|          | 0/114 [00:00<?, ?it/s]
| Epoch: 50 | Train Loss: 0.318 | Train Acc: 99.56% | Val. Loss: 0.347 | Val. Acc: 9
6.67% | Val. Precision: 0.978 | Val. Recall: 0.967 | Val. F1-score: 0.964 |
```

In [64]:

```
test_loss, test_acc, test_prec, test_rec, test_f1 = evaluate(cancermodel, test_dataloa

print(f'| Test. Loss: {test_loss:.3f} | Test. Acc: {test_acc*100:.2f}% | Test. Precisi
```

```
| Test. Loss: 0.346 | Test. Acc: 96.43% | Test. Precision: 0.976 | Test. Recall: 0.96
4 | Test. F1-score: 0.962 |
```