#Problem Description

Histopathologic Cancer Detection

This is a binary image classification task aimed at identifying the presence or absence of metastatic cancer in small image patches extracted from larger digital pathology scans.

The objective of this project is to predict the labels for these images. A positive label indicates that the central 32x32 pixel region of a patch contains at least one pixel of tumor tissue.

```
#import libraries
import os
import numpy as np
import pandas as pd
from skimage.io import imread
import torch
import torch.nn as nn
import torchvision.transforms as transforms
from torch.utils.data import Dataset,DataLoader
from torchvision.io import read_image
import os
import cv2
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import accuracy_score,roc_auc_score
import time
import copy
from tqdm import tqdm_notebook as tqdm
from PIL import Image
import random
random.seed(42)

import warnings
warnings.filterwarnings("ignore", category=DeprecationWarning)
warnings.filterwarnings("ignore", category=UserWarning)
warnings.filterwarnings("ignore", category=FutureWarning)

#setting environment and parameters
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
num_epochs = 30 #30
batch_size = 256 #32 #256
num_classes = 2
learning_rate = 0.01

# to load data
import os
import pandas as pd
from google.colab import drive
drive.mount('/content/drive', force_remount=True)
```

```
dir_main =
'/content/drive/MyDrive/CuBoulder/DTSA5511DeepLearning/wk3/CancerData'
path = dir_main + '/train/'
annotation_file = dir_main + '/train_labels.csv'
test_path=dir_main + '/test/'

num_train_files = len(os.listdir(path))
num_test_files = len(os.listdir(test_path))

train = pd.read_csv(annotation_file)
sub = pd.read_csv(dir_main + '/sample_submission.csv')
```

```
Mounted at /content/drive
```

Dataset Description

This dataset contains a large number of small pathology images to classify. Files are named with an image id. The train_labels.csv file provides the ground truth for the images in the train folder.

There are two folders where small pathology images are contained. There are 220025 tif files in the train folder and 57458 tif files in the test folder respectively. The total size of the both folders combined is about 7.76GB.

Explortary Data Analysis

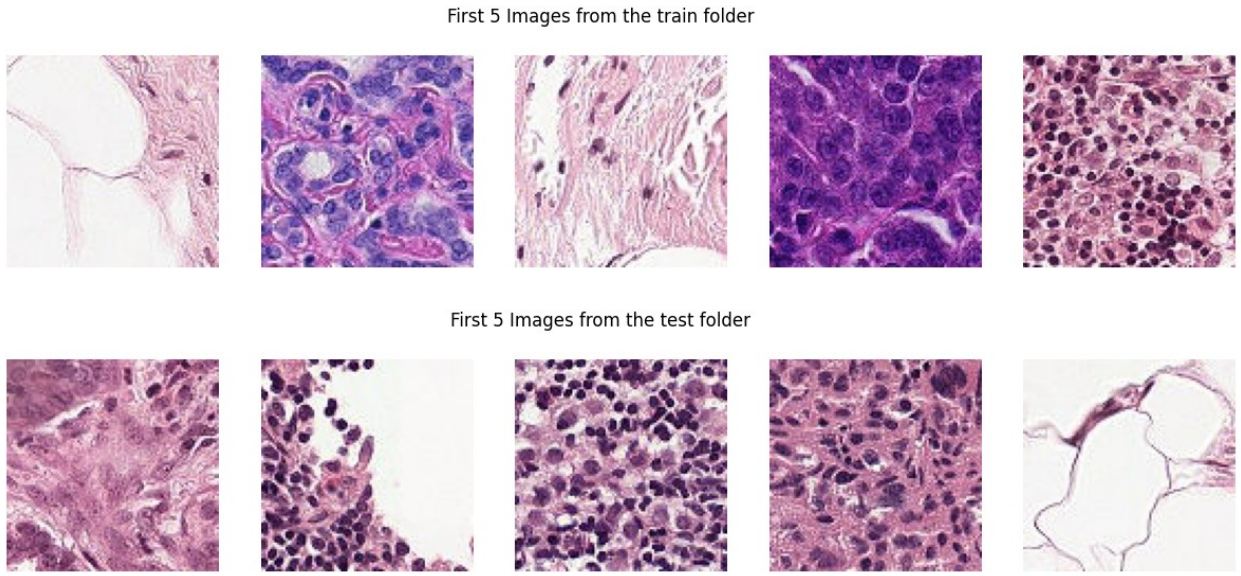
Show a few visualizations like histograms. Describe any data cleaning procedures. Based on your EDA, what is your plan of analysis?

show some cancer images from both folders

The first 5 images of train images and test images are shown below.

```
# show the first 5 images in the train folder
train_files = os.listdir(path)
fig, ax = plt.subplots(1, 5, figsize=(15, 3))

for i in range(5):
    image_path = os.path.join(path, train_files[i])
    image = plt.imread(image_path)
    ax[i].imshow(image)
    ax[i].axis('off')

plt.suptitle('First 5 Images from the train folder')
plt.show()
```

```
# show the first 5 images in the test folder
train_files = os.listdir(test_path)
fig, ax = plt.subplots(1, 5, figsize=(15, 3))

for i in range(5):
    image_path = os.path.join(test_path, train_files[i])
    image = plt.imread(image_path)
    ax[i].imshow(image)
    ax[i].axis('off')

plt.suptitle('First 5 Images from the test folder')
plt.show()
```



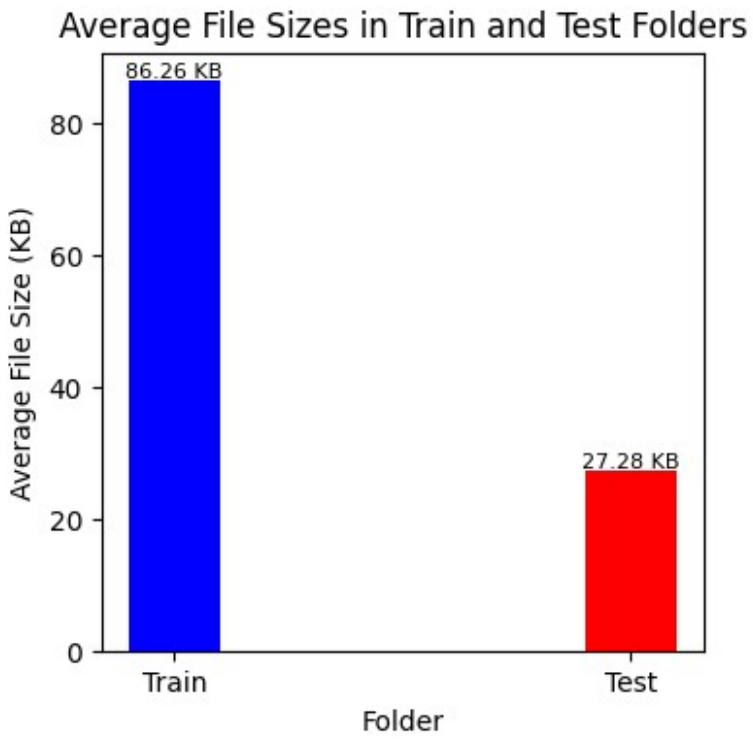
check on the image file size.

I checked the size of the cancer images to see if there is anything to learn from it. Based on the bar chart below, all training images have the same size of 86 KB, while all test images have a size of 27 KB.

I was surprised to find that the training images differ in size from the testing images. It is unclear why they are different. It may be important to keep this in mind to determine if there is a need to treat the testing images differently.

```
import matplotlib.pyplot as plt
import os

# Calculate file sizes for each folder
train_files = os.listdir(path)
train_sizes = [os.path.getsize(os.path.join(path, f)) / 1024 for f in
train_files]
```

```
test_files = os.listdir(test_path)
test_sizes = [os.path.getsize(os.path.join(test_path, f)) / 1024 for f
in test_files]

# Create a figure and axes
fig, ax = plt.subplots(figsize=(4,4))

# Calculate the average file sizes
avg_train_size = sum(train_sizes) / len(train_sizes)
avg_test_size = sum(test_sizes) / len(test_sizes)

# Plot the average file sizes as a bar chart
train_bar = ax.bar(['Train'], [avg_train_size], width=0.2, color='b')
test_bar = ax.bar(['Test'], [avg_test_size], width=0.2, color='r')

# Add the average file size value on each bar
ax.text(0, avg_train_size, f"{avg_train_size:.2f} KB", ha='center',
va='bottom', fontsize=8)
ax.text(1, avg_test_size, f"{avg_test_size:.2f} KB", ha='center',
va='bottom', fontsize=8)

# Add labels and title
ax.set_xlabel("Folder")
ax.set_ylabel("Average File Size (KB)")
ax.set_title("Average File Sizes in Train and Test Folders")

# Show the plot
plt.show()
```



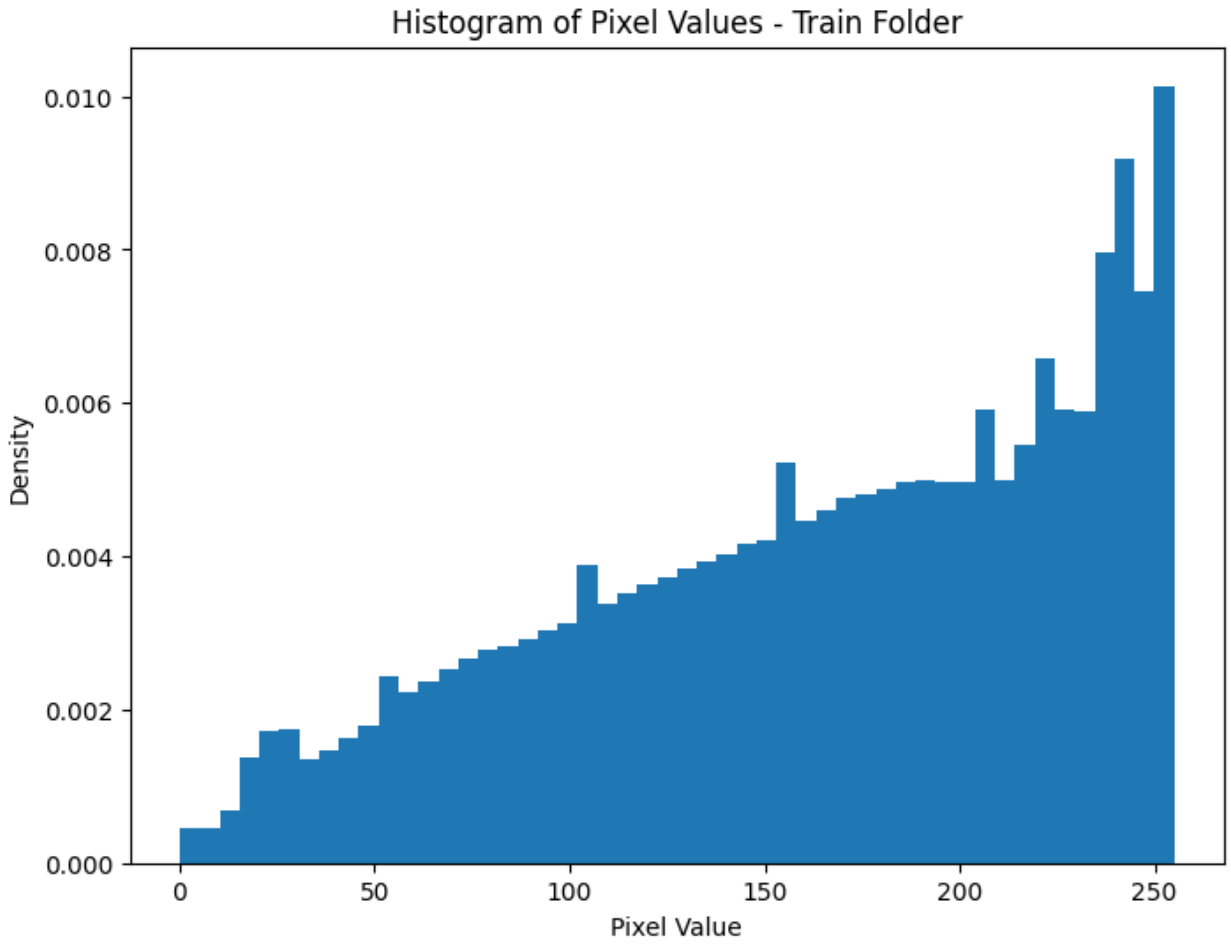
Analyzing the pixel value of the images

The following charts show the distribution of pixel values for the images in both the train and test folders. The mean pixel value is 159.91 for train images and 162.34 for test images. All the images tend to be light in color, generally having a light-colored background.

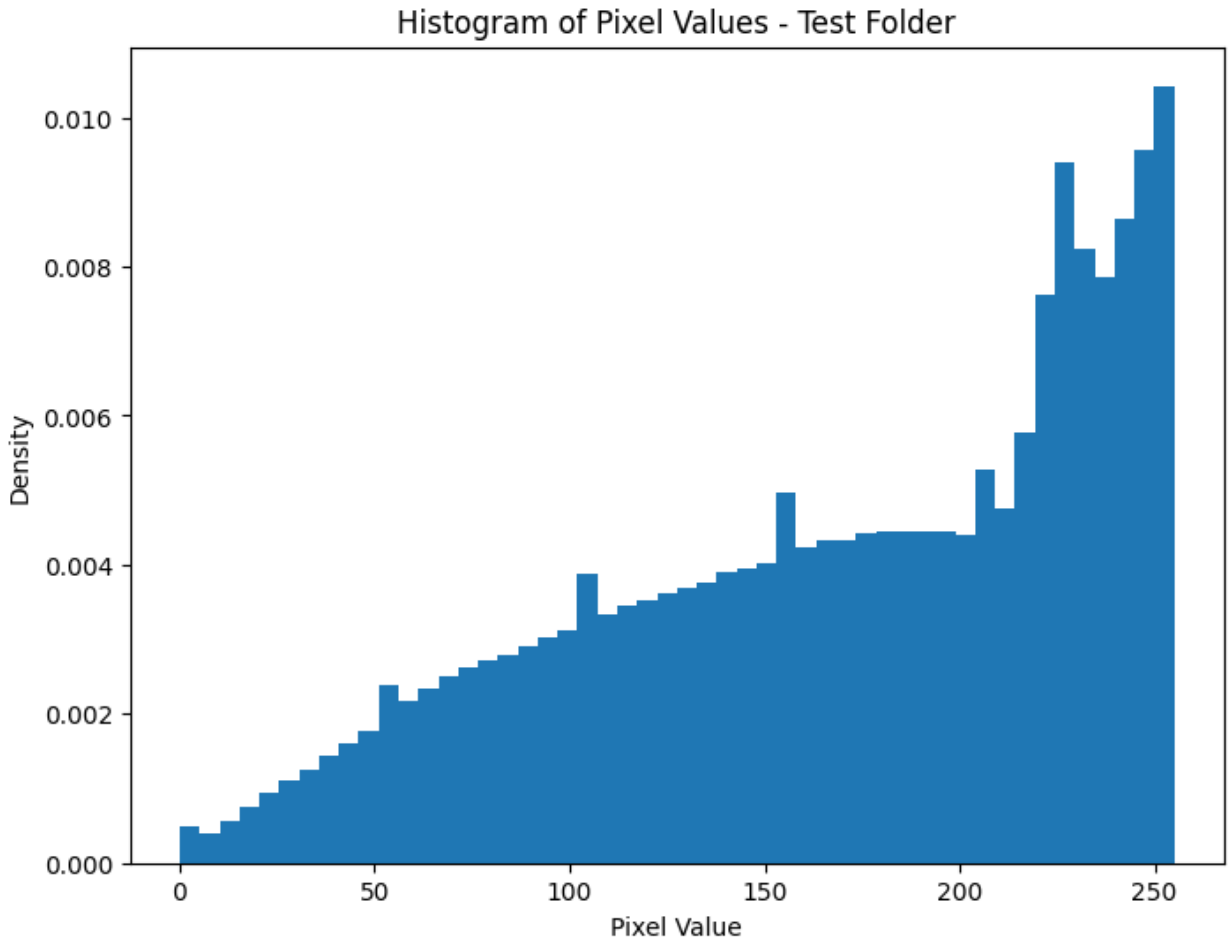
```
# Pixel Value Distribution: Compute summary statistics (mean, median,
standard deviation) of pixel values.

import os
import numpy as np
from PIL import Image

#dir_main = '/content/drive/MyDrive/CU
Boulder/DTSA5511DeepLearning/wk3/CancerData'
#train = os.path.join(dir_main, 'train')
#test_path = os.path.join(dir_main, 'test')

def compute_pixel_stats(folder_path, num_files=100):
    all_pixel_values = []
    tif_files = [f for f in os.listdir(folder_path) if
f.endswith('.tif')]

    # Randomly select 1,000 files
    selected_files = random.sample(tif_files, min(num_files,
len(tif_files)))
```

```
    for filename in selected_files:
        image_path = os.path.join(folder_path, filename)
        image = Image.open(image_path)
        pixel_values = np.array(image).flatten()
        all_pixel_values.extend(pixel_values)

    pixel_mean = np.mean(all_pixel_values)
    pixel_median = np.median(all_pixel_values)
    pixel_std = np.std(all_pixel_values)

    return all_pixel_values, pixel_mean, pixel_median, pixel_std

# Compute statistics and plot histogram for the train folder
train_pixel_values, train_mean, train_median, train_std =
compute_pixel_stats(path)
print("Train Folder:")
print(f"Mean: {train_mean:.2f}")
print(f"Median: {train_median:.2f}")
print(f"Standard Deviation: {train_std:.2f}")

plt.figure(figsize=(8, 6))
plt.hist(train_pixel_values, bins=50, density=True)
plt.title('Histogram of Pixel Values - Train Folder')
plt.xlabel('Pixel Value')
plt.ylabel('Density')
plt.show()

# Compute statistics and plot histogram for the test folder
test_pixel_values, test_mean, test_median, test_std =
compute_pixel_stats(test_path)
print("\nTest Folder:")
print(f"Mean: {test_mean:.2f}")
print(f"Median: {test_median:.2f}")
print(f"Standard Deviation: {test_std:.2f}")

plt.figure(figsize=(8, 6))
plt.hist(test_pixel_values, bins=50, density=True)
plt.title('Histogram of Pixel Values - Test Folder')
plt.xlabel('Pixel Value')
plt.ylabel('Density')
plt.show()
```

```
Train Folder:
Mean: 164.44
Median: 175.00
Standard Deviation: 65.98
```




```
Test Folder:
Mean: 168.94
Median: 180.00
Standard Deviation: 65.27
```



Reducing training and testing data size

The original dataset is quite large, with over 200,000 training images, which likely exceeds the capacity of the available infrastructure. Therefore, I have reduced the dataset to 4,000 training images and 5,000 testing images.

Since the CSV file contains labels for images that are not part of the selected subset, the following code identifies the labels from the CSV file for the images chosen for training and testing.

```
# divide train data into train set and validation set

train_set, val_set = train_test_split(train, stratify=train['label'],
test_size=0.05)
print(len(train_set), len(val_set))

# match dataframe of train_set and val_set based on actual files
existed in folder

# check train set if files do exist or not
existing_count = 0
```

```
missing_count = 0
existing_index = []
for i, v in enumerate(train_set.itertuples(index=True)):
  image_path = os.path.join(path, v.id + '.tif')
  if os.path.exists(image_path):
    existing_count += 1
    existing_index.append(v.Index)
    #print(f"Image {image_path} exists.")
  else:
    missing_count += 1
    #print(f"Image {image_path} does not exist.")
print(f"Number of existing images: {existing_count}")
print(f"Number of missing images: {missing_count}")
train_set = train_set.loc[existing_index]  # update train_set_sample
with only the ones that their files exist

# check val set if files do exist or not
existing_count = 0
missing_count = 0
existing_index = []
for i, v in enumerate(val_set.itertuples(index=True)):
  image_path = os.path.join(path, v.id + '.tif')
  if os.path.exists(image_path):
    existing_count += 1
    existing_index.append(v.Index)
    #print(f"Image {image_path} exists.")
  else:
    missing_count += 1
    #print(f"Image {image_path} does not exist.")
print(f"Number of existing images: {existing_count}")
print(f"Number of missing images: {missing_count}")
val_set = val_set.loc[existing_index]  # update train_set_sample with
only the ones that their files exist

# check test set if files do exist or not
existing_count = 0
missing_count = 0
existing_index = []
for i, v in enumerate(sub.itertuples(index=True)):
  image_path = os.path.join(test_path, v.id + '.tif')
  if os.path.exists(image_path):
    existing_count += 1
    existing_index.append(v.Index)
    #print(f"Image {image_path} exists.")
  else:
    missing_count += 1
    #print(f"Image {image_path} does not exist.")
print(f"Number of existing images: {existing_count}")
print(f"Number of missing images: {missing_count}")
```

```
sub = sub.loc[existing_index]  # update train_set_sample with only the
ones that their files exist

print(f"train_set.shape: {train_set.shape}")
print(f"val_set.shape: {val_set.shape}")
print(f"sub.shape: {sub.shape}")
######################### to randomly select a subset of the data
sample_size_train = 4000
sample_size_val = 150
sample_size_sub = 5000
random_numbers = random.sample(range(train_set.shape[0]),
sample_size_train)
# the train of files based on the random numbers
train_set = train_set.iloc[random_numbers]
# check validation set if the randomly selected image exists or not;
remove them if not existed
random_numbers = random.sample(range(val_set.shape[0]),
sample_size_val)
# the train of files based on the random numbers
val_set = val_set.iloc[random_numbers]
# check test set if the randomly selected image exists or not; remove
them if not existed
random_numbers = random.sample(range(sub.shape[0]), sample_size_sub)
# the train of files based on the random numbers
sub = sub.iloc[random_numbers]
print(f"after sampling train_set.shape: {train_set.shape}")
print(f"after sampling val_set.shape: {val_set.shape}")
print(f"after sampling sub.shape: {sub.shape}")
```

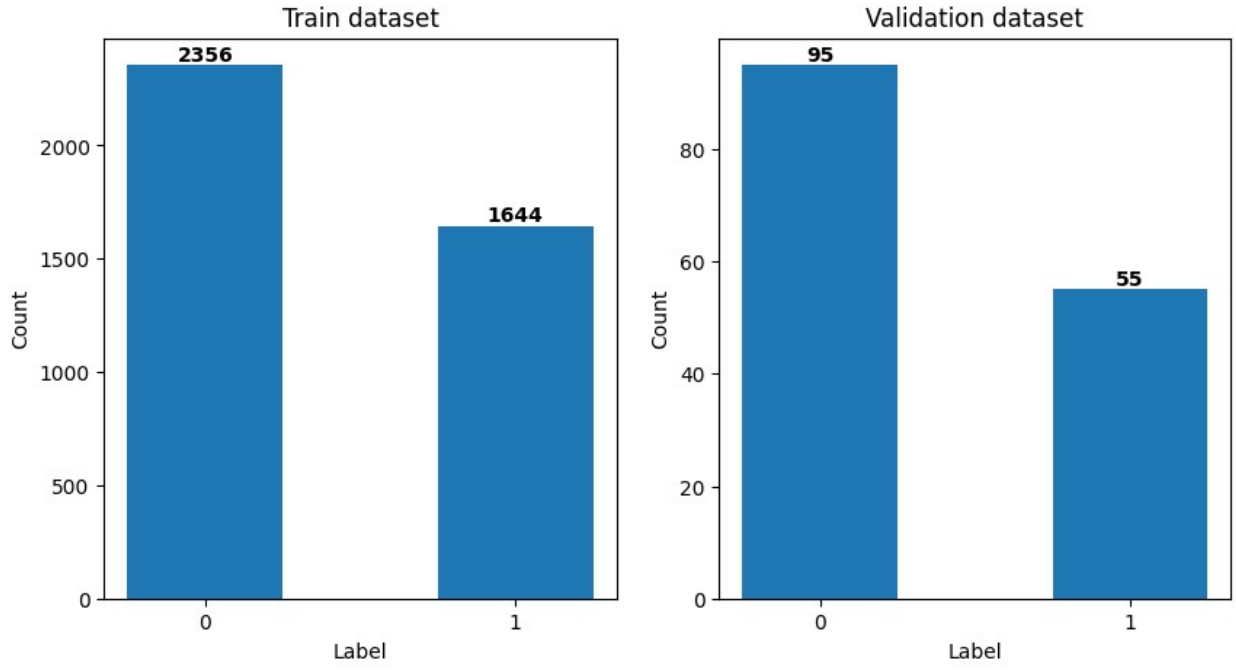
```
209023 11002
Number of existing images: 4742
Number of missing images: 204281
Number of existing images: 258
Number of missing images: 10744
Number of existing images: 5000
Number of missing images: 52458
train_set.shape: (4742, 2)
val_set.shape: (258, 2)
sub.shape: (5000, 2)
after sampling train_set.shape: (4000, 2)
after sampling val_set.shape: (150, 2)
after sampling sub.shape: (5000, 2)
```

checking on the balance of the label

The following bar charts show the breakdown between cancer images and non-cancer images for both the train and test sets. The train set has 2,356 negative labeled images and 1,644 positive images, while the test set has 95 negative labeled images and 55 positive images. The balance between negative and positive labels seems to be adequate for building a model.

```
import matplotlib.pyplot as plt

# check the balance of the label for train and validation dataset
fig, ax = plt.subplots(1, 2, figsize=(10, 5))
# plot train label bar chat
train_labels = train_set['label'].value_counts()
for i, v in enumerate(train_labels.values):
    ax[0].text(i, v, str(v), ha='center', va='bottom',
fontweight='bold')
ax[0].bar([0, 1], train_labels.values, width=0.5)
ax[0].set_title('Train dataset')
ax[0].set_xlabel('Label')
ax[0].set_ylabel('Count')
ax[0].set_xticks([0, 1])
ax[0].set_xticklabels(['0', '1'])
# plot validation label bar chat
val_labels = val_set['label'].value_counts()
for i, v in enumerate(val_labels.values):
    ax[1].text(i, v, str(v), ha='center', va='bottom',
fontweight='bold')
ax[1].bar([0, 1], val_labels.values, width=0.5)
ax[1].set_title('Validation dataset')
ax[1].set_xlabel('Label')
ax[1].set_ylabel('Count')
ax[1].set_xticks([0, 1])
ax[1].set_xticklabels(['0', '1'])
```

```
[Text(0, 0, '0'), Text(1, 0, '1')]
```



Model Architecture (25 pts)

Model Building and Training

This problem involves analyzing images, so using a convolutional neural network (CNN) for machine learning is appropriate. I will experiment with different CNN architectures to build a model that predicts whether images contain cancer.

I will start with a simple CNN model and then iterate, adding more advanced layers and complex architectures to find the best-performing model.

The neural network layers I will use to build this model include:

- Convolution layer
- MaxPooling layer
- Fully Connected Linear layer
- ReLU layer
- BatchNorm layer
- Dropout layer

I have chosen an epoch count of 30 and a batch size of 256.

```
# define customDataset object

class CancerDataset(Dataset):

    def __init__(self, df_data, data_dir = './', transform=None):
        super().__init__()
        self.df = df_data.values
        self.data_dir = data_dir
        self.transform = transform

    def __len__(self):
        return len(self.df)

    def __getitem__(self, index):
        img_name,label = self.df[index]
        img_path = os.path.join(self.data_dir, img_name + '.tif')
        image = cv2.imread(img_path)
        if self.transform is not None:
            image = self.transform(image)
        return image, label
```

Build version 1 model

The first model is relatively simple, with just two convolutional layers and a fully connected layer that includes linear, ReLU, BatchNorm1d, and Dropout layers. There are no MaxPooling layers in the convolutional layers.

This model will serve as a baseline to test how well it performs with a simpler and straightforward structure.

```
class CNN(nn.Module):
    def __init__(self):
        super(CNN,self).__init__()
        self.conv1 = nn.Sequential(
                        nn.Conv2d(3, 32, 3, stride=1, padding=1),
                        nn.BatchNorm2d(32),
                        nn.ReLU(inplace=True))#,
        self.conv2 = nn.Sequential(
                        nn.Conv2d(32, 64, 3, stride=1, padding=1),
                        nn.BatchNorm2d(64),
                        nn.ReLU(inplace=True))#,
        self.fc=nn.Sequential(
                nn.Linear(64*96*96, 128),
                nn.ReLU(inplace=True),
                nn.BatchNorm1d(128),
                nn.Dropout(0.4),
                nn.Linear(128, num_classes))
    def forward(self,x):
        x=self.conv1(x)
        x=self.conv2(x)
        x=x.view(x.shape[0],-1)
        x=self.fc(x)
        return x

# version one transform
transform_train = transforms.Compose([
    transforms.ToPILImage(),
    #transforms.Resize((224, 224)),
    transforms.ToTensor(),
    transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))
])

transform_val = transforms.Compose([transforms.ToPILImage(),
                                    transforms.ToTensor(),
                                    transforms.Normalize(mean=[0.5, 0.5,
0.5],std=[0.5, 0.5, 0.5])])

transform_test = transforms.Compose([transforms.ToPILImage(),
                                    transforms.ToTensor(),
                                    transforms.Normalize(mean=[0.5, 0.5,
0.5],std=[0.5, 0.5, 0.5])])

model = CNN().to(device)
print(model)

CNN(
  (conv1): Sequential(
```

```
    (0): Conv2d(3, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1))
    (1): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (2): ReLU(inplace=True)
  )
  (conv2): Sequential(
    (0): Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1))
    (1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (2): ReLU(inplace=True)
  )
  (fc): Sequential(
    (0): Linear(in_features=589824, out_features=128, bias=True)
    (1): ReLU(inplace=True)
    (2): BatchNorm1d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (3): Dropout(p=0.4, inplace=False)
    (4): Linear(in_features=128, out_features=2, bias=True)
  )
)
```

```
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import DataLoader
from sklearn.metrics import accuracy_score,roc_auc_score
import torchvision.transforms as transforms

train_auc = []
val_auc = []
train_auc_epoch = []
val_auc_epoch = []
best_acc = 0.0
min_loss = np.Inf

# Load the dataset
train_dataset = CancerDataset(df_data=train_set, data_dir=path,
transform=transform_train)
val_dataset = CancerDataset(df_data=val_set, data_dir=path,
transform=transform_train)
test_dataset = CancerDataset(df_data=sub, data_dir=test_path,
transform=transform_test)
print(f"batch_size: {batch_size}")
# Create DataLoaders
train_loader = DataLoader(train_dataset, batch_size=batch_size,
shuffle=True, num_workers=4)
val_loader = DataLoader(val_dataset, batch_size=batch_size,
shuffle=False, num_workers=4)
```

```
test_dataloader = DataLoader(test_dataset, batch_size=batch_size,
shuffle=False)

# Train the model
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model = CNN().to(device)
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)
for epoch in range(num_epochs):
    train_loss = 0.0
    val_loss = 0.0
    running_loss = 0.0

    # Training
    model.train()
    print(train_loader)
    for i, (inputs, labels) in enumerate(tqdm(train_loader,
total=int(len(train_loader)))):

        inputs = inputs.to(device)
        labels = labels.to(device)
        # Forward pass
        outputs = model(inputs)
        loss = criterion(outputs, labels)
        # Backward and optimize
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
        # Loss and accuracy
        train_loss += loss.item()
        y_actual = labels.data.cpu().numpy()
        y_pred = outputs[:,-1].detach().cpu().numpy()
        train_auc.append(roc_auc_score(y_actual, y_pred))

    print(f'Epoch [{epoch+1}/{num_epochs}], Training Loss: {train_loss
/ len(train_loader):.4f}')

    # Validation
    model.eval()
    correct = 0
    total = 0
    with torch.no_grad():
        for inputs, labels in (tqdm(val_loader,
total=int(len(val_loader)))):
            inputs, labels = inputs.to(device), labels.to(device)
            outputs = model(inputs)
            # Loss and accuracy
            loss = criterion(outputs, labels)
            val_loss += loss.item()
            y_actual = labels.data.cpu().numpy()
```

```
            y_pred = outputs[:,-1].detach().cpu().numpy()
            val_auc.append(roc_auc_score(y_actual, y_pred))
            _, predicted = torch.max(outputs.data, 1)
            total += labels.size(0)
            correct += (predicted == labels).sum().item()

        #auc
        training_auc = np.mean(train_auc)
        validation_auc = np.mean(val_auc)
        train_auc_epoch.append(training_auc)
        val_auc_epoch.append(validation_auc)

        # Updating best validation accuracy
        if best_acc < validation_auc:
            best_acc = validation_auc

        # Saving best model
        if min_loss >= val_loss:
            torch.save(model.state_dict(), 'best_model.pt')
            min_loss = val_loss

    print(f'Epoch [{epoch+1}/{num_epochs}], Validation Accuracy: {100
* correct / total}%')
```

```
batch_size: 256
<torch.utils.data.dataloader.DataLoader object at 0x7e06f028e9b0>
```

{"model_id":"61236ad12a7142a4b44a6c8c68eb7f4d","version_major":2,"vers
ion_minor":0}

```
Epoch [1/30], Training Loss: 0.5446
```

{"model_id":"ebf3e3ec070847e49c49c1e3fb624d92","version_major":2,"vers
ion_minor":0}

```
Epoch [1/30], Validation Accuracy: 46.666666666666664%
<torch.utils.data.dataloader.DataLoader object at 0x7e06f028e9b0>
```

{"model_id":"1006e9bf11bb4f97bd6d03640ae7fcea","version_major":2,"vers
ion_minor":0}

```
Epoch [2/30], Training Loss: 0.4526
```

{"model_id":"05a7c167d57a449d8f5f6d1e8f13afa5","version_major":2,"vers
ion_minor":0}

```
Epoch [2/30], Validation Accuracy: 62.666666666666664%
<torch.utils.data.dataloader.DataLoader object at 0x7e06f028e9b0>
```

{"model_id":"abc2d8715cf6421da23247c9cc5b5917","version_major":2,"vers
ion_minor":0}

```
Epoch [3/30], Training Loss: 0.4229

{"model_id":"411b3a1b93b2427a91dc0963d9bceba5","version_major":2,"vers
ion_minor":0}

Epoch [3/30], Validation Accuracy: 62.0%
<torch.utils.data.dataloader.DataLoader object at 0x7e06f028e9b0>

{"model_id":"3b3ecf45082e47778369f9b4c3735ca8","version_major":2,"vers
ion_minor":0}

Epoch [4/30], Training Loss: 0.4040

{"model_id":"6b50f3395e304a13bf1cab1bd71b2982","version_major":2,"vers
ion_minor":0}

Epoch [4/30], Validation Accuracy: 79.33333333333333%
<torch.utils.data.dataloader.DataLoader object at 0x7e06f028e9b0>

{"model_id":"fed62ebf76c84febaa7c4720f164549b","version_major":2,"vers
ion_minor":0}

Epoch [5/30], Training Loss: 0.3850

{"model_id":"06e5257040844cdf88b0c67df9f1e161","version_major":2,"vers
ion_minor":0}

Epoch [5/30], Validation Accuracy: 63.333333333333336%
<torch.utils.data.dataloader.DataLoader object at 0x7e06f028e9b0>

{"model_id":"81076bd548fc4d15bfded06a0aea5200","version_major":2,"vers
ion_minor":0}

Epoch [6/30], Training Loss: 0.3537

{"model_id":"c06de8166b8c47cb9fd27e4ba553ce9b","version_major":2,"vers
ion_minor":0}

Epoch [6/30], Validation Accuracy: 78.0%
<torch.utils.data.dataloader.DataLoader object at 0x7e06f028e9b0>

{"model_id":"182a920814c14707b32c2aa2eb7d9bc3","version_major":2,"vers
ion_minor":0}

Epoch [7/30], Training Loss: 0.3182

{"model_id":"40374d59e1ea445d945ad9b64ccc02c5","version_major":2,"vers
ion_minor":0}

Epoch [7/30], Validation Accuracy: 82.0%
<torch.utils.data.dataloader.DataLoader object at 0x7e06f028e9b0>
```

{"model_id":"7d5649412196 47f7b885bb57c4de4b1a","version_major":2,"version_minor":0}

```
Epoch [8/30], Training Loss: 0.2510
```

{"model_id":"e4b118334aac46c284793da48e2e009c","version_major":2,"version_minor":0}

```
Epoch [8/30], Validation Accuracy: 74.66666666666667%
<torch.utils.data.dataloader.DataLoader object at 0x7e06f028e9b0>
```

{"model_id":"6b2018eb072a4369ba23ac5d664fb2c4","version_major":2,"version_minor":0}

```
Epoch [9/30], Training Loss: 0.1863
```

{"model_id":"7f79ce610fbc4d2a88192044ed77cc82","version_major":2,"version_minor":0}

```
Epoch [9/30], Validation Accuracy: 75.33333333333333%
<torch.utils.data.dataloader.DataLoader object at 0x7e06f028e9b0>
```

{"model_id":"a75d4f151d074b868a7e7abc18dbdb86","version_major":2,"version_minor":0}

```
Epoch [10/30], Training Loss: 0.1438
```

{"model_id":"7a51d15520fe43f0824374c13b820795","version_major":2,"version_minor":0}

```
Epoch [10/30], Validation Accuracy: 72.0%
<torch.utils.data.dataloader.DataLoader object at 0x7e06f028e9b0>
```

{"model_id":"406c3f1fbe8d417f8a8ac54351d0bea8","version_major":2,"version_minor":0}

```
Epoch [11/30], Training Loss: 0.0993
```

{"model_id":"c841284fdb1241febf46a64723a43474","version_major":2,"version_minor":0}

```
Epoch [11/30], Validation Accuracy: 76.66666666666667%
<torch.utils.data.dataloader.DataLoader object at 0x7e06f028e9b0>
```

{"model_id":"40a7258922ad44e7abdb7b74649c8325","version_major":2,"version_minor":0}

```
Epoch [12/30], Training Loss: 0.0654
```

{"model_id":"4d769b984d5c4fd98987530842644b2a","version_major":2,"version_minor":0}

```
Epoch [12/30], Validation Accuracy: 78.66666666666667%
<torch.utils.data.dataloader.DataLoader object at 0x7e06f028e9b0>

{"model_id":"32b12d39a40f49878d85a2beb437a4ab","version_major":2,"vers
ion_minor":0}

Epoch [13/30], Training Loss: 0.0373

{"model_id":"8ec760c169f54bf4a3b591b6a033e47f","version_major":2,"vers
ion_minor":0}

Epoch [13/30], Validation Accuracy: 76.66666666666667%
<torch.utils.data.dataloader.DataLoader object at 0x7e06f028e9b0>

{"model_id":"f611fedcb81a439d8239815fd5ce2448","version_major":2,"vers
ion_minor":0}

Epoch [14/30], Training Loss: 0.0267

{"model_id":"552d44cb268e4263af08c13af63b79ec","version_major":2,"vers
ion_minor":0}

Epoch [14/30], Validation Accuracy: 74.66666666666667%
<torch.utils.data.dataloader.DataLoader object at 0x7e06f028e9b0>

{"model_id":"bf15ef3fa36d4c44884d9e13483bb349","version_major":2,"vers
ion_minor":0}

Epoch [15/30], Training Loss: 0.0285

{"model_id":"a7cbd8ed369c42ae829a3aad003d7acd","version_major":2,"vers
ion_minor":0}

Epoch [15/30], Validation Accuracy: 76.66666666666667%
<torch.utils.data.dataloader.DataLoader object at 0x7e06f028e9b0>

{"model_id":"ed7913b43e77423c9623a032a6acc6b7","version_major":2,"vers
ion_minor":0}

Epoch [16/30], Training Loss: 0.0184

{"model_id":"9946b9d8189243bcaae8de196121ec55","version_major":2,"vers
ion_minor":0}

Epoch [16/30], Validation Accuracy: 77.33333333333333%
<torch.utils.data.dataloader.DataLoader object at 0x7e06f028e9b0>

{"model_id":"467df947cc4149558acf4412086ecfcc","version_major":2,"vers
ion_minor":0}

Epoch [17/30], Training Loss: 0.0210
```

{"model_id":"05dfd8391f9540a0b7fcd43847239fcf","version_major":2,"version_minor":0}

```
Epoch [17/30], Validation Accuracy: 76.66666666666667%
<torch.utils.data.dataloader.DataLoader object at 0x7e06f028e9b0>
```

{"model_id":"8ed9c85366574f2999fd82be35b28c1e","version_major":2,"version_minor":0}

```
Epoch [18/30], Training Loss: 0.0095
```

{"model_id":"d6f895014bf64b1286c0a5d7fb06f4db","version_major":2,"version_minor":0}

```
Epoch [18/30], Validation Accuracy: 76.0%
<torch.utils.data.dataloader.DataLoader object at 0x7e06f028e9b0>
```

{"model_id":"ba776216d36642b1b90e5275744ed902","version_major":2,"version_minor":0}

```
Epoch [19/30], Training Loss: 0.0083
```

{"model_id":"97d878183d2c44adbe5ab89b823fdda7","version_major":2,"version_minor":0}

```
Epoch [19/30], Validation Accuracy: 75.33333333333333%
<torch.utils.data.dataloader.DataLoader object at 0x7e06f028e9b0>
```

{"model_id":"8bf758638ec747fc86c7b4ab88da8729","version_major":2,"version_minor":0}

```
Epoch [20/30], Training Loss: 0.0066
```

{"model_id":"f9aa05716cf34812aed76e2adb46412c","version_major":2,"version_minor":0}

```
Epoch [20/30], Validation Accuracy: 76.66666666666667%
<torch.utils.data.dataloader.DataLoader object at 0x7e06f028e9b0>
```

{"model_id":"7489e2f616be4113a9552cb9a2845c09","version_major":2,"version_minor":0}

```
Epoch [21/30], Training Loss: 0.0099
```

{"model_id":"049befb8bf184ddb817c4c9758991d95","version_major":2,"version_minor":0}

```
Epoch [21/30], Validation Accuracy: 79.33333333333333%
<torch.utils.data.dataloader.DataLoader object at 0x7e06f028e9b0>
```

{"model_id":"4488442bac79401f929cc8b5cb64b17c","version_major":2,"version_minor":0}

```
Epoch [22/30], Training Loss: 0.0068
```


```
{"model_id":"a32e1eaddff64c7a8f1dbb4aa8277f36","version_major":2,"vers
ion_minor":0}
```


```
Epoch [22/30], Validation Accuracy: 69.33333333333333%
<torch.utils.data.dataloader.DataLoader object at 0x7e06f028e9b0>
```


```
{"model_id":"91828f6f8718477a83e61c8963a852b7","version_major":2,"vers
ion_minor":0}
```


```
Epoch [23/30], Training Loss: 0.0056
```


```
{"model_id":"6d61a5a774b54eb9b3524160cfa855f0","version_major":2,"vers
ion_minor":0}
```


```
Epoch [23/30], Validation Accuracy: 78.0%
<torch.utils.data.dataloader.DataLoader object at 0x7e06f028e9b0>
```


```
{"model_id":"5f04e3f865d246488f27ccaec227db96","version_major":2,"vers
ion_minor":0}
```


```
Epoch [24/30], Training Loss: 0.0029
```


```
{"model_id":"34614318b71945b39dc43b13ebba5831","version_major":2,"vers
ion_minor":0}
```


```
Epoch [24/30], Validation Accuracy: 74.0%
<torch.utils.data.dataloader.DataLoader object at 0x7e06f028e9b0>
```


```
{"model_id":"5a33a2885ef447c3bb2b2edcc4742b87","version_major":2,"vers
ion_minor":0}
```


```
Epoch [25/30], Training Loss: 0.0025
```


```
{"model_id":"bc5d5fcc607a4adc83c0f911b8e3d43e","version_major":2,"vers
ion_minor":0}
```


```
Epoch [25/30], Validation Accuracy: 74.66666666666667%
<torch.utils.data.dataloader.DataLoader object at 0x7e06f028e9b0>
```


```
{"model_id":"8bce6cc1242d4600adbee76098121e7c","version_major":2,"vers
ion_minor":0}
```


```
Epoch [26/30], Training Loss: 0.0024
```


```
{"model_id":"a4810005483f48b78fa839ab2ed560d4","version_major":2,"vers
ion_minor":0}
```


```
Epoch [26/30], Validation Accuracy: 72.66666666666667%
<torch.utils.data.dataloader.DataLoader object at 0x7e06f028e9b0>
```

{"model_id":"334001d9072e4aff91311bc7a732c744","version_major":2,"version_minor":0}

```
Epoch [27/30], Training Loss: 0.0015
```

{"model_id":"71e2c6f1af1d4140a5c28ac15d5e80e8","version_major":2,"version_minor":0}

```
Epoch [27/30], Validation Accuracy: 75.33333333333333%
<torch.utils.data.dataloader.DataLoader object at 0x7e06f028e9b0>
```

{"model_id":"532341cbc2234bc9a887c9460448d59b","version_major":2,"version_minor":0}

```
Epoch [28/30], Training Loss: 0.0015
```

{"model_id":"db935b852842402392d0c7752de1d0d3","version_major":2,"version_minor":0}

```
Epoch [28/30], Validation Accuracy: 74.66666666666667%
<torch.utils.data.dataloader.DataLoader object at 0x7e06f028e9b0>
```

{"model_id":"7e8c7d15cda14f478de0306bcbb572d2","version_major":2,"version_minor":0}

```
Epoch [29/30], Training Loss: 0.0020
```

{"model_id":"1f6bfa13d5c3412391fec49354701975","version_major":2,"version_minor":0}

```
Epoch [29/30], Validation Accuracy: 77.33333333333333%
<torch.utils.data.dataloader.DataLoader object at 0x7e06f028e9b0>
```

{"model_id":"8573add209e34a789d8667d39a5d59df","version_major":2,"version_minor":0}

```
Epoch [30/30], Training Loss: 0.0017
```

{"model_id":"08fa8d0f340a4f999fe1e2a10a0a1500","version_major":2,"version_minor":0}

```
Epoch [30/30], Validation Accuracy: 77.33333333333333%
```

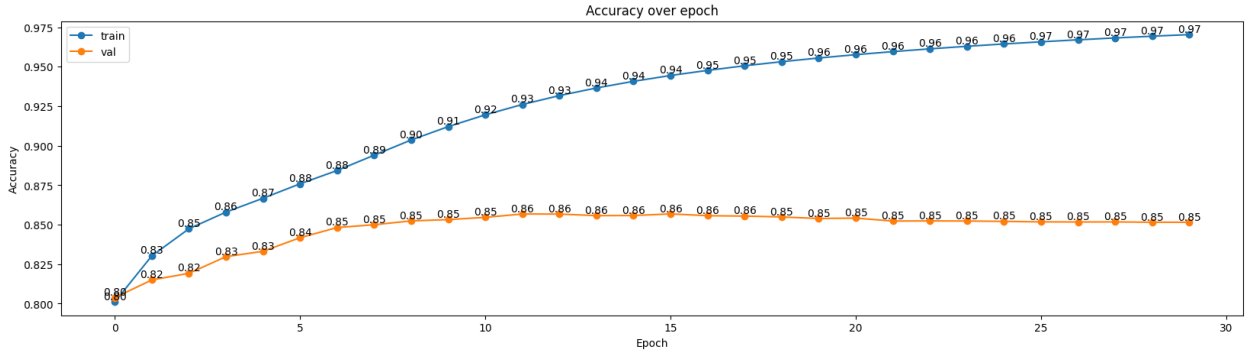
Results and Analysis

Based on the accuracy rate plot below, both the training and validation accuracy are progressing relatively well. However, while the training accuracy is advancing towards 97%, the validation accuracy is stuck at around 85%. This discrepancy could be due to overfitting, as the model is learning the training data too well.

For the next version, I will try a more complex CNN architecture with additional regularization parameters. Since 30 epochs seem to work fine, it may not be necessary to train for more epochs.

```
print(f"first version of model")
plt.figure(figsize=(20,5))
plt.plot(train_auc_epoch, '-o', label="train")
plt.plot(val_auc_epoch, '-o', label="val")
# Adding value labels
for i, value in enumerate(train_auc_epoch):
    plt.text(i, value, f"{value:.2f}", ha='center', va='bottom')
for i, value in enumerate(val_auc_epoch):
    plt.text(i, value, f"{value:.2f}", ha='center', va='bottom')
plt.xlabel("Epoch")
plt.ylabel("Accuracy")
plt.title("Accuracy over epoch")
plt.legend()
```

```
first version of model

<matplotlib.legend.Legend at 0x7e05d632b190>
```




```
import pickle
# Save the lists to a file
with open('model_metrics1.pkl', 'wb') as f:
    pickle.dump([train_auc, val_auc, train_auc_epoch, val_auc_epoch],
f)

import pickle
with open('model_metrics1.pkl', 'rb') as f:
    train_auc, val_auc, train_auc_epoch, val_auc_epoch =
pickle.load(f)
```

Building version 2 model

Although the first version is not bad, there is definitely room for improvement. In Version 2, I will add more convolutional layers. Additionally, 2D MaxPooling layers will be added after each convolutional layer to help mitigate the overfitting issue.

```
# version 2 CNN

class CNN(nn.Module):
    def __init__(self):
        super(CNN,self).__init__()

        self.conv1 = nn.Sequential(
                        nn.Conv2d(3, 32, 3, stride=1, padding=1),
                        nn.BatchNorm2d(32),
                        nn.ReLU(inplace=True),
                        nn.MaxPool2d(2,2))
        self.conv2 = nn.Sequential(
                        nn.Conv2d(32, 64, 3, stride=1, padding=1),
                        nn.BatchNorm2d(64),
                        nn.ReLU(inplace=True),
                        nn.MaxPool2d(2,2))
        self.conv3 = nn.Sequential(
                        nn.Conv2d(64, 128, 3, stride=1, padding=1),
                        nn.BatchNorm2d(128),
                        nn.ReLU(inplace=True),
                        nn.MaxPool2d(2,2))
        self.conv4 = nn.Sequential(
                        nn.Conv2d(128, 256, 3, stride=1, padding=1),
                        nn.BatchNorm2d(256),
                        nn.ReLU(inplace=True),
                        nn.MaxPool2d(2,2))
        self.conv5 = nn.Sequential(
                        nn.Conv2d(256, 512, 3, stride=1, padding=1),
                        nn.BatchNorm2d(512),
                        nn.ReLU(inplace=True),
                        nn.MaxPool2d(2,2))
        self.fc=nn.Sequential(
                nn.Linear(512*3*3, 256),
                nn.ReLU(inplace=True),
                nn.BatchNorm1d(256),
                nn.Dropout(0.4),
                nn.Linear(256, num_classes))

    def forward(self,x):
        x=self.conv1(x)
        x=self.conv2(x)
        x=self.conv3(x)
        x=self.conv4(x)
        x=self.conv5(x)
        x=x.view(x.shape[0],-1)
        x=self.fc(x)
        return x

# version two transform (same as version one)
# Define the transforms
```

```
transform_train = transforms.Compose([
    transforms.ToPILImage(),
    #transforms.Resize((224, 224)),
    transforms.ToTensor(),
    transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))
])

transform_val = transforms.Compose([transforms.ToPILImage(),
                                    transforms.ToTensor(),
                                    transforms.Normalize(mean=[0.5, 0.5,
0.5],std=[0.5, 0.5, 0.5])])

transform_test = transforms.Compose([transforms.ToPILImage(),
                                    transforms.ToTensor(),
                                    transforms.Normalize(mean=[0.5, 0.5,
0.5],std=[0.5, 0.5, 0.5])])

model = CNN().to(device)
print(model)
```

```
CNN(
  (conv1): Sequential(
    (0): Conv2d(3, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1))
    (1): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (2): ReLU(inplace=True)
    (3): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
  )
  (conv2): Sequential(
    (0): Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1))
    (1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (2): ReLU(inplace=True)
    (3): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
  )
  (conv3): Sequential(
    (0): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1))
    (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (2): ReLU(inplace=True)
    (3): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
  )
  (conv4): Sequential(
    (0): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1),
```

```
padding=(1, 1))
    (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (2): ReLU(inplace=True)
    (3): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
  )
  (conv5): Sequential(
    (0): Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1))
    (1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (2): ReLU(inplace=True)
    (3): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
  )
  (fc): Sequential(
    (0): Linear(in_features=4608, out_features=256, bias=True)
    (1): ReLU(inplace=True)
    (2): BatchNorm1d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (3): Dropout(p=0.4, inplace=False)
    (4): Linear(in_features=256, out_features=2, bias=True)
  )
)

import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import DataLoader
from sklearn.metrics import accuracy_score,roc_auc_score
import torchvision.transforms as transforms

train_auc = []
val_auc = []
train_auc_epoch = []
val_auc_epoch = []
best_acc = 0.0
min_loss = np.Inf

# Load the dataset
train_dataset = CancerDataset(df_data=train_set, data_dir=path,
transform=transform_train)
val_dataset = CancerDataset(df_data=val_set, data_dir=path,
transform=transform_train)
test_dataset = CancerDataset(df_data=sub, data_dir=test_path,
transform=transform_test)
print(f"batch_size: {batch_size}")
# Create DataLoaders
train_loader = DataLoader(train_dataset, batch_size=batch_size,
```

```
shuffle=True, num_workers=4)
val_loader = DataLoader(val_dataset, batch_size=batch_size,
shuffle=False, num_workers=4)
test_dataloader = DataLoader(test_dataset, batch_size=batch_size,
shuffle=False)

# Train the model
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model = CNN().to(device)
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)

for epoch in range(num_epochs):
    train_loss = 0.0
    val_loss = 0.0
    running_loss = 0.0

    # Training
    model.train()
    print(train_loader)
    for i, (inputs, labels) in enumerate(tqdm(train_loader,
total=int(len(train_loader)))):
        inputs = inputs.to(device)
        labels = labels.to(device)
        # Forward pass
        outputs = model(inputs)
        loss = criterion(outputs, labels)
        # Backward and optimize
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
        # Loss and accuracy
        train_loss += loss.item()
        y_actual = labels.data.cpu().numpy()
        y_pred = outputs[:,-1].detach().cpu().numpy()
        train_auc.append(roc_auc_score(y_actual, y_pred))

    print(f'Epoch [{epoch+1}/{num_epochs}], Training Loss: {train_loss
/ len(train_loader):.4f}')

    # Validation
    model.eval()
    correct = 0
    total = 0
    with torch.no_grad():
        for inputs, labels in (tqdm(val_loader,
total=int(len(val_loader)))):
            inputs, labels = inputs.to(device), labels.to(device)
            outputs = model(inputs)
            # Loss and accuracy
```

```
            loss = criterion(outputs, labels)
            val_loss += loss.item()
            y_actual = labels.data.cpu().numpy()
            y_pred = outputs[:,-1].detach().cpu().numpy()
            val_auc.append(roc_auc_score(y_actual, y_pred))
            _, predicted = torch.max(outputs.data, 1)
            total += labels.size(0)
            correct += (predicted == labels).sum().item()
        #auc
        training_auc = np.mean(train_auc)
        validation_auc = np.mean(val_auc)
        train_auc_epoch.append(training_auc)
        val_auc_epoch.append(validation_auc)
        # Updating best validation accuracy
        if best_acc < validation_auc:
            best_acc = validation_auc
        # Saving best model
        if min_loss >= val_loss:
            torch.save(model.state_dict(), 'best_model.pt')
            min_loss = val_loss
    print(f'Epoch [{epoch+1}/{num_epochs}], Validation Accuracy: {100
* correct / total}%')
```

```
batch_size: 256
<torch.utils.data.dataloader.DataLoader object at 0x7e06a91bfa30>
```

{"model_id":"913ede0be6854d8a9726a5ecd7c30f35","version_major":2,"vers
ion_minor":0}

```
Epoch [1/30], Training Loss: 0.5474
```

{"model_id":"2135b9df270d4a879330b1f53450613c","version_major":2,"vers
ion_minor":0}

```
Epoch [1/30], Validation Accuracy: 59.333333333333336%
<torch.utils.data.dataloader.DataLoader object at 0x7e06a91bfa30>
```

{"model_id":"76217fcf500d42bc8a088ccd1b3e28b1","version_major":2,"vers
ion_minor":0}

```
Epoch [2/30], Training Loss: 0.4275
```

{"model_id":"8b9bab1ed19a4c7e932d7c9cc75a1747","version_major":2,"vers
ion_minor":0}

```
Epoch [2/30], Validation Accuracy: 68.66666666666667%
<torch.utils.data.dataloader.DataLoader object at 0x7e06a91bfa30>
```

{"model_id":"9a47d93484d0469cb3fdedd7c1812875","version_major":2,"vers
ion_minor":0}

```
Epoch [3/30], Training Loss: 0.3944
```

{"model_id":"674e38b38e0d435c8bfd50ca31c793bf","version_major":2,"version_minor":0}

```
Epoch [3/30], Validation Accuracy: 79.33333333333333%
<torch.utils.data.dataloader.DataLoader object at 0x7e06a91bfa30>
```

{"model_id":"c3bcba06afe9422a8349d1a2fc3e0ee3","version_major":2,"version_minor":0}

```
Epoch [4/30], Training Loss: 0.3849
```

{"model_id":"bc1e14408289423891de6bdf4fe0f304","version_major":2,"version_minor":0}

```
Epoch [4/30], Validation Accuracy: 79.33333333333333%
<torch.utils.data.dataloader.DataLoader object at 0x7e06a91bfa30>
```

{"model_id":"34fe6e8783c0453dbaf29dbf32279bfc","version_major":2,"version_minor":0}

```
Epoch [5/30], Training Loss: 0.3750
```

{"model_id":"886cf139e84845599228114e46450035","version_major":2,"version_minor":0}

```
Epoch [5/30], Validation Accuracy: 77.33333333333333%
<torch.utils.data.dataloader.DataLoader object at 0x7e06a91bfa30>
```

{"model_id":"42c3361619e44266810ea2b820499c5a","version_major":2,"version_minor":0}

```
Epoch [6/30], Training Loss: 0.3666
```

{"model_id":"1ac722e5d82b45faa25a049d4555a6af","version_major":2,"version_minor":0}

```
Epoch [6/30], Validation Accuracy: 80.0%
<torch.utils.data.dataloader.DataLoader object at 0x7e06a91bfa30>
```

{"model_id":"c9f23f1901474741a55ef26a11872594","version_major":2,"version_minor":0}

```
Epoch [7/30], Training Loss: 0.3482
```

{"model_id":"c6b1765f3e794f1181b801499edbd662","version_major":2,"version_minor":0}

```
Epoch [7/30], Validation Accuracy: 82.0%
<torch.utils.data.dataloader.DataLoader object at 0x7e06a91bfa30>
```

{"model_id":"16c31a1b4e454f44b3ef9de1e7cd6623","version_major":2,"version_minor":0}

```
Epoch [8/30], Training Loss: 0.3244

{"model_id":"4d94f649075f43b794b39b45fb9702bd","version_major":2,"vers
ion_minor":0}

Epoch [8/30], Validation Accuracy: 73.33333333333333%
<torch.utils.data.dataloader.DataLoader object at 0x7e06a91bfa30>

{"model_id":"fa5ede817cf84350a651775784b69832","version_major":2,"vers
ion_minor":0}

Epoch [9/30], Training Loss: 0.3291

{"model_id":"2a564aec14c44d6b8b3f38ab7828701b","version_major":2,"vers
ion_minor":0}

Epoch [9/30], Validation Accuracy: 73.33333333333333%
<torch.utils.data.dataloader.DataLoader object at 0x7e06a91bfa30>

{"model_id":"750ec7d51aa94d658528bc40a2ca1306","version_major":2,"vers
ion_minor":0}

Epoch [10/30], Training Loss: 0.3157

{"model_id":"f3cba37d13f4439ba03d3dd1a6f950c6","version_major":2,"vers
ion_minor":0}

Epoch [10/30], Validation Accuracy: 80.0%
<torch.utils.data.dataloader.DataLoader object at 0x7e06a91bfa30>

{"model_id":"d4c447ac3aa04b2ea6159d76aafde9b1","version_major":2,"vers
ion_minor":0}

Epoch [11/30], Training Loss: 0.2804

{"model_id":"62467923c06d47688a66d5d1de39d209","version_major":2,"vers
ion_minor":0}

Epoch [11/30], Validation Accuracy: 80.0%
<torch.utils.data.dataloader.DataLoader object at 0x7e06a91bfa30>

{"model_id":"cec5f86168ca46e48496a44898f624bb","version_major":2,"vers
ion_minor":0}

Epoch [12/30], Training Loss: 0.2741

{"model_id":"19af03c5ba79422a91bcf42a420f5181","version_major":2,"vers
ion_minor":0}

Epoch [12/30], Validation Accuracy: 80.0%
<torch.utils.data.dataloader.DataLoader object at 0x7e06a91bfa30>
```

{"model_id":"2746b1c5b38245f48820f7a578338a61","version_major":2,"version_minor":0}

```
Epoch [13/30], Training Loss: 0.2491
```

{"model_id":"cd5ffc9ee5a24aa4aa5a13b56e344486","version_major":2,"version_minor":0}

```
Epoch [13/30], Validation Accuracy: 81.33333333333333%
<torch.utils.data.dataloader.DataLoader object at 0x7e06a91bfa30>
```

{"model_id":"0a2138331c694c158d0e31110cf86a09","version_major":2,"version_minor":0}

```
Epoch [14/30], Training Loss: 0.2188
```

{"model_id":"255c33c690f6415082f635562ed4c1fb","version_major":2,"version_minor":0}

```
Epoch [14/30], Validation Accuracy: 82.0%
<torch.utils.data.dataloader.DataLoader object at 0x7e06a91bfa30>
```

{"model_id":"c8af4bfa5ed647e4a66756bce63af0a1","version_major":2,"version_minor":0}

```
Epoch [15/30], Training Loss: 0.1613
```

{"model_id":"cbc74569893e4f6bb15bcf5b0f413d11","version_major":2,"version_minor":0}

```
Epoch [15/30], Validation Accuracy: 88.0%
<torch.utils.data.dataloader.DataLoader object at 0x7e06a91bfa30>
```

{"model_id":"938fc64110f54be8baf2e9ce996a1ba9","version_major":2,"version_minor":0}

```
Epoch [16/30], Training Loss: 0.1367
```

{"model_id":"71ec87336f77448a94a9b434be16154f","version_major":2,"version_minor":0}

```
Epoch [16/30], Validation Accuracy: 82.0%
<torch.utils.data.dataloader.DataLoader object at 0x7e06a91bfa30>
```

{"model_id":"8e1f342ff7f04b738540b34c48a00520","version_major":2,"version_minor":0}

```
Epoch [17/30], Training Loss: 0.0829
```

{"model_id":"50c8ce66954042149f33104d620d62db","version_major":2,"version_minor":0}

```
Epoch [17/30], Validation Accuracy: 80.0%
<torch.utils.data.dataloader.DataLoader object at 0x7e06a91bfa30>

{"model_id":"44b7d2c9720a4ae4b0e1016f1f95ac4d","version_major":2,"vers
ion_minor":0}

Epoch [18/30], Training Loss: 0.0795

{"model_id":"42fdf80476af48579aa245b0fea4a4ea","version_major":2,"vers
ion_minor":0}

Epoch [18/30], Validation Accuracy: 80.0%
<torch.utils.data.dataloader.DataLoader object at 0x7e06a91bfa30>

{"model_id":"64e87f41057540bc8e5d1b29cbf832a0","version_major":2,"vers
ion_minor":0}

Epoch [19/30], Training Loss: 0.0438

{"model_id":"dfee3f43d16c42ca9fce7ce6b9b63a9c","version_major":2,"vers
ion_minor":0}

Epoch [19/30], Validation Accuracy: 82.66666666666667%
<torch.utils.data.dataloader.DataLoader object at 0x7e06a91bfa30>

{"model_id":"dbb84740ea604f87b780d77cb338818d","version_major":2,"vers
ion_minor":0}

Epoch [20/30], Training Loss: 0.0376

{"model_id":"33ecddb93e214b67a9bad5a27026672d","version_major":2,"vers
ion_minor":0}

Epoch [20/30], Validation Accuracy: 77.33333333333333%
<torch.utils.data.dataloader.DataLoader object at 0x7e06a91bfa30>

{"model_id":"84db07c7aef1488db5af874d1b76b329","version_major":2,"vers
ion_minor":0}

Epoch [21/30], Training Loss: 0.0232

{"model_id":"0b64e5418baa47fba942a76c602f9251","version_major":2,"vers
ion_minor":0}

Epoch [21/30], Validation Accuracy: 82.66666666666667%
<torch.utils.data.dataloader.DataLoader object at 0x7e06a91bfa30>

{"model_id":"030f1552bb1148ada44a876046f94605","version_major":2,"vers
ion_minor":0}

Epoch [22/30], Training Loss: 0.0158
```

{"model_id":"a87abcb3cd73466ba4ee0aea89e17e58","version_major":2,"version_minor":0}

```
Epoch [22/30], Validation Accuracy: 78.0%
<torch.utils.data.dataloader.DataLoader object at 0x7e06a91bfa30>
```

{"model_id":"85d4d85714e34474a2b9c0d32b92a942","version_major":2,"version_minor":0}

```
Epoch [23/30], Training Loss: 0.0088
```

{"model_id":"723f1d4d22bd408c8b856056fe703348","version_major":2,"version_minor":0}

```
Epoch [23/30], Validation Accuracy: 84.66666666666667%
<torch.utils.data.dataloader.DataLoader object at 0x7e06a91bfa30>
```

{"model_id":"3b49946aafc64ee58b1b41b92973d021","version_major":2,"version_minor":0}

```
Epoch [24/30], Training Loss: 0.0034
```

{"model_id":"8e1a389f51154a62bce3a5e92a362021","version_major":2,"version_minor":0}

```
Epoch [24/30], Validation Accuracy: 84.0%
<torch.utils.data.dataloader.DataLoader object at 0x7e06a91bfa30>
```

{"model_id":"761e5d1fa2de4846990dc6a56e1baa93","version_major":2,"version_minor":0}

```
Epoch [25/30], Training Loss: 0.0013
```

{"model_id":"9bc56ef723b747ada71a6b02c9ab1766","version_major":2,"version_minor":0}

```
Epoch [25/30], Validation Accuracy: 84.0%
<torch.utils.data.dataloader.DataLoader object at 0x7e06a91bfa30>
```

{"model_id":"b32d5e7caace4aa78536e20de51861b7","version_major":2,"version_minor":0}

```
Epoch [26/30], Training Loss: 0.0010
```

{"model_id":"f9866195edaa4481b8e0bd4131fcfb1c","version_major":2,"version_minor":0}

```
Epoch [26/30], Validation Accuracy: 82.66666666666667%
<torch.utils.data.dataloader.DataLoader object at 0x7e06a91bfa30>
```

{"model_id":"181621b0c828499b9dacef9d437cd0b4","version_major":2,"version_minor":0}

```
Epoch [27/30], Training Loss: 0.0007

{"model_id":"940944c93818422eb63fb5cfad3875f7","version_major":2,"vers
ion_minor":0}

Epoch [27/30], Validation Accuracy: 85.33333333333333%
<torch.utils.data.dataloader.DataLoader object at 0x7e06a91bfa30>

{"model_id":"b5aec9f47d8a4721bb0b84ffa8aa39e8","version_major":2,"vers
ion_minor":0}

Epoch [28/30], Training Loss: 0.0005

{"model_id":"9877fe62b2c64827b14d82b7530418f8","version_major":2,"vers
ion_minor":0}

Epoch [28/30], Validation Accuracy: 84.0%
<torch.utils.data.dataloader.DataLoader object at 0x7e06a91bfa30>

{"model_id":"b2a360e115ed458ba74c54809a0e94b8","version_major":2,"vers
ion_minor":0}

Epoch [29/30], Training Loss: 0.0004

{"model_id":"226d893b192042a3a7759b6fd0cad3a1","version_major":2,"vers
ion_minor":0}

Epoch [29/30], Validation Accuracy: 83.33333333333333%
<torch.utils.data.dataloader.DataLoader object at 0x7e06a91bfa30>

{"model_id":"4777da7426c74c2d83337f01adca56c3","version_major":2,"vers
ion_minor":0}

Epoch [30/30], Training Loss: 0.0003

{"model_id":"1cb6961a517845d5977c4e4d990103ff","version_major":2,"vers
ion_minor":0}

Epoch [30/30], Validation Accuracy: 84.66666666666667%
```

Results and Analysis

The Version 2 model performs better than Version 1. The training set accuracy reaches about 96%, while the validation accuracy remains around 89%, showing a slight improvement over Version 1. The advanced CNN model has further improved the training accuracy, but the improvement in validation accuracy is limited. The overfitting issue is certainly addressed.

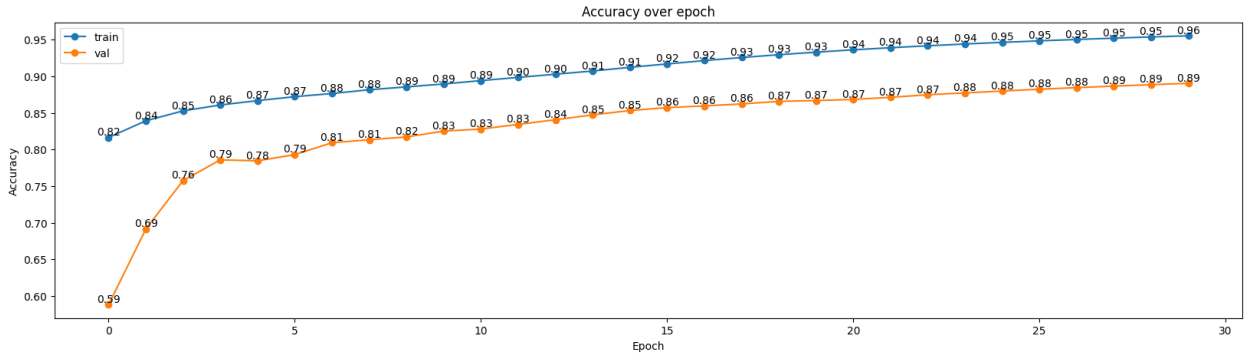
It is unclear if further increasing the complexity of the CNN architecture will help. For the next version, I will focus on more data augmentation to create a greater variety of images to see if the model can learn better.

Additionally, it may be possible to optimize the learning rate by increasing it slightly. This could speed up the learning process without compromising the quality of the machine learning.

```
print(f"Second version of model")
plt.figure(figsize=(20,5))
plt.plot(train_auc_epoch, '-o', label="train")
plt.plot(val_auc_epoch, '-o', label="val")
# Adding value labels
for i, value in enumerate(train_auc_epoch):
    plt.text(i, value, f"{value:.2f}", ha='center', va='bottom')
for i, value in enumerate(val_auc_epoch):
    plt.text(i, value, f"{value:.2f}", ha='center', va='bottom')
plt.xlabel("Epoch")
plt.ylabel("Accuracy")
plt.title("Accuracy over epoch")
plt.legend()
```

```
Second version of model
```

```
<matplotlib.legend.Legend at 0x7e07076b7880>
```




```
import pickle
# Save the lists to a file
with open('model_metrics2.pkl', 'wb') as f:
    pickle.dump([train_auc, val_auc, train_auc_epoch, val_auc_epoch],
f)

import pickle
with open('model_metrics2.pkl', 'rb') as f:
    train_auc, val_auc, train_auc_epoch, val_auc_epoch =
pickle.load(f)
```

Building version 3 model

In this version, I am introducing more complex data augmentation by adding random flipping and rotation to the training data. I will incorporate RandomHorizontalFlip, RandomVerticalFlip, and RandomRotation to generate more variety from the existing images. This augmentation could help the model better learn the differentiation between positive and negative cases.

Additionally, the learning rate is increased to 0.01 to try to speed up the learning process.

```
# Version 3 CNN

class CNN(nn.Module):
    def __init__(self):
        super(CNN,self).__init__()

        self.conv1 = nn.Sequential(
                        nn.Conv2d(3, 32, 3, stride=1, padding=1),
                        nn.BatchNorm2d(32),
                        nn.ReLU(inplace=True),
                        nn.MaxPool2d(2,2))
        self.conv2 = nn.Sequential(
                        nn.Conv2d(32, 64, 3, stride=1, padding=1),
                        nn.BatchNorm2d(64),
                        nn.ReLU(inplace=True),
                        nn.MaxPool2d(2,2))
        self.conv3 = nn.Sequential(
                        nn.Conv2d(64, 128, 3, stride=1, padding=1),
                        nn.BatchNorm2d(128),
                        nn.ReLU(inplace=True),
                        nn.MaxPool2d(2,2))
        self.conv4 = nn.Sequential(
                        nn.Conv2d(128, 256, 3, stride=1, padding=1),
                        nn.BatchNorm2d(256),
                        nn.ReLU(inplace=True),
                        nn.MaxPool2d(2,2))
        self.conv5 = nn.Sequential(
                        nn.Conv2d(256, 512, 3, stride=1, padding=1),
                        nn.BatchNorm2d(512),
                        nn.ReLU(inplace=True),
                        nn.MaxPool2d(2,2))
        self.fc=nn.Sequential(
                nn.Linear(512*3*3, 256),
                nn.ReLU(inplace=True),
                nn.BatchNorm1d(256),
                nn.Dropout(0.4),
                nn.Linear(256, num_classes))

    def forward(self,x):
        x=self.conv1(x)
        x=self.conv2(x)
        x=self.conv3(x)
        x=self.conv4(x)
        x=self.conv5(x)
        x=x.view(x.shape[0],-1)
        x=self.fc(x)
        return x
```

```
# Version 3
transform_train = transforms.Compose([transforms.ToPILImage(),
                                      transforms.RandomHorizontalFlip(),
                                      transforms.RandomVerticalFlip(),
                                      transforms.RandomRotation(20),
                                      transforms.ToTensor(),
                                      transforms.Normalize(mean=[0.5, 0.5,
0.5],std=[0.5, 0.5, 0.5])])

transform_val = transforms.Compose([transforms.ToPILImage(),
                                      transforms.ToTensor(),
                                      transforms.Normalize(mean=[0.5, 0.5,
0.5],std=[0.5, 0.5, 0.5])])

transform_test = transforms.Compose([transforms.ToPILImage(),
                                      transforms.ToTensor(),
                                      transforms.Normalize(mean=[0.5, 0.5,
0.5],std=[0.5, 0.5, 0.5])])

model = CNN().to(device)
print(model)

CNN(
  (conv1): Sequential(
    (0): Conv2d(3, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1))
    (1): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (2): ReLU(inplace=True)
    (3): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
  )
  (conv2): Sequential(
    (0): Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1))
    (1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (2): ReLU(inplace=True)
    (3): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
  )
  (conv3): Sequential(
    (0): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1))
    (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (2): ReLU(inplace=True)
    (3): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
  )
```

```
  (conv4): Sequential(
    (0): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1))
    (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (2): ReLU(inplace=True)
    (3): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
  )
  (conv5): Sequential(
    (0): Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1))
    (1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (2): ReLU(inplace=True)
    (3): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
  )
  (fc): Sequential(
    (0): Linear(in_features=4608, out_features=256, bias=True)
    (1): ReLU(inplace=True)
    (2): BatchNorm1d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (3): Dropout(p=0.4, inplace=False)
    (4): Linear(in_features=256, out_features=2, bias=True)
  )
)

# version 3 - learning rate set to 0.01

learning_rate = 0.01
train_auc = []
val_auc = []
train_auc_epoch = []
val_auc_epoch = []
best_acc = 0.0
min_loss = np.Inf

# Load the dataset
train_dataset = CancerDataset(df_data=train_set, data_dir=path,
transform=transform_train)
val_dataset = CancerDataset(df_data=val_set, data_dir=path,
transform=transform_train)
test_dataset = CancerDataset(df_data=sub, data_dir=test_path,
transform=transform_test)
print(f"batch_size: {batch_size}")
# Create DataLoaders
train_loader = DataLoader(train_dataset, batch_size=batch_size,
shuffle=True, num_workers=4)
val_loader = DataLoader(val_dataset, batch_size=batch_size,
```

```
shuffle=False, num_workers=4)
test_dataloader = DataLoader(test_dataset, batch_size=batch_size,
shuffle=False)

# Train the model
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model = CNN().to(device)
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=learning_rate)

for epoch in range(num_epochs):
    train_loss = 0.0
    val_loss = 0.0
    running_loss = 0.0

    # Training
    model.train()
    print(train_loader)
    for i, (inputs, labels) in enumerate(tqdm(train_loader,
total=int(len(train_loader)))):
        inputs = inputs.to(device)
        labels = labels.to(device)
        # Forward pass
        outputs = model(inputs)
        loss = criterion(outputs, labels)
        # Backward and optimize
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
        # Loss and accuracy
        train_loss += loss.item()
        y_actual = labels.data.cpu().numpy()
        y_pred = outputs[:,-1].detach().cpu().numpy()
        train_auc.append(roc_auc_score(y_actual, y_pred))

    print(f'Epoch [{epoch+1}/{num_epochs}], Training Loss: {train_loss
/ len(train_loader):.4f}')

    # Validation
    model.eval()
    correct = 0
    total = 0
    with torch.no_grad():
        for inputs, labels in (tqdm(val_loader,
total=int(len(val_loader)))):
            inputs, labels = inputs.to(device), labels.to(device)
            outputs = model(inputs)
            # Loss and accuracy
            loss = criterion(outputs, labels)
            val_loss += loss.item()
```

```
            y_actual = labels.data.cpu().numpy()
            y_pred = outputs[:,-1].detach().cpu().numpy()
            val_auc.append(roc_auc_score(y_actual, y_pred))
            _, predicted = torch.max(outputs.data, 1)
            total += labels.size(0)
            correct += (predicted == labels).sum().item()
        #auc
        training_auc = np.mean(train_auc)
        validation_auc = np.mean(val_auc)
        train_auc_epoch.append(training_auc)
        val_auc_epoch.append(validation_auc)
        # Updating best validation accuracy
        if best_acc < validation_auc:
            best_acc = validation_auc
        # Saving best model
        if min_loss >= val_loss:
            torch.save(model.state_dict(), 'best_model.pt')
            min_loss = val_loss
    print(f'Epoch [{epoch+1}/{num_epochs}], Validation Accuracy: {100
* correct / total}%')
```

```
batch_size: 256
<torch.utils.data.dataloader.DataLoader object at 0x7e0707bb7760>
```


```
{"model_id":"93c1ed301cb4457fb38094aca517167e","version_major":2,"vers
ion_minor":0}
```


```
Epoch [1/30], Training Loss: 0.6825
```


```
{"model_id":"0b43a2fa891f4ee78eded0e996c14feb","version_major":2,"vers
ion_minor":0}
```


```
Epoch [1/30], Validation Accuracy: 64.0%
<torch.utils.data.dataloader.DataLoader object at 0x7e0707bb7760>
```


```
{"model_id":"3d0cfa6cf21446dba14d84e29b8f87ba","version_major":2,"vers
ion_minor":0}
```


```
Epoch [2/30], Training Loss: 0.5117
```


```
{"model_id":"b3b46f131a5d4703bb894a139a1abfb3","version_major":2,"vers
ion_minor":0}
```


```
Epoch [2/30], Validation Accuracy: 69.33333333333333%
<torch.utils.data.dataloader.DataLoader object at 0x7e0707bb7760>
```


```
{"model_id":"38a9a78d0551466881bc0807b71aa2e8","version_major":2,"vers
ion_minor":0}
```


```
Epoch [3/30], Training Loss: 0.4787
```

{"model_id":"e2e537d1e7e644c2963bed80bf6653d7","version_major":2,"version_minor":0}

```
Epoch [3/30], Validation Accuracy: 74.66666666666667%
<torch.utils.data.dataloader.DataLoader object at 0x7e0707bb7760>
```

{"model_id":"968fa391fdbd448db017ff51b2dbe1ba","version_major":2,"version_minor":0}

```
Epoch [4/30], Training Loss: 0.4518
```

{"model_id":"33aca00c56754631ad04b0b7686a545f","version_major":2,"version_minor":0}

```
Epoch [4/30], Validation Accuracy: 76.0%
<torch.utils.data.dataloader.DataLoader object at 0x7e0707bb7760>
```

{"model_id":"a5189e053937401a84c7a1e2a39125af","version_major":2,"version_minor":0}

```
Epoch [5/30], Training Loss: 0.4434
```

{"model_id":"02133931e41648a9ac42d9064178900b","version_major":2,"version_minor":0}

```
Epoch [5/30], Validation Accuracy: 76.66666666666667%
<torch.utils.data.dataloader.DataLoader object at 0x7e0707bb7760>
```

{"model_id":"0ef4e8fa747b4010b2cfb2674b88babf","version_major":2,"version_minor":0}

```
Epoch [6/30], Training Loss: 0.4217
```

{"model_id":"6f33f3fd70714f5d9b079316b98a4c1e","version_major":2,"version_minor":0}

```
Epoch [6/30], Validation Accuracy: 78.0%
<torch.utils.data.dataloader.DataLoader object at 0x7e0707bb7760>
```

{"model_id":"cd9882ba31c4492cbbb8ceb40a69c01c","version_major":2,"version_minor":0}

```
Epoch [7/30], Training Loss: 0.4189
```

{"model_id":"1477311585414a2aacfda361c4fd0961","version_major":2,"version_minor":0}

```
Epoch [7/30], Validation Accuracy: 86.0%
<torch.utils.data.dataloader.DataLoader object at 0x7e0707bb7760>
```

{"model_id":"9bb5d26ca9ed4b4eb230d3de4a1cf544","version_major":2,"version_minor":0}

```
Epoch [8/30], Training Loss: 0.4086

{"model_id":"b4dec406b3424cbab49336ae52501b34","version_major":2,"vers
ion_minor":0}

Epoch [8/30], Validation Accuracy: 80.0%
<torch.utils.data.dataloader.DataLoader object at 0x7e0707bb7760>

{"model_id":"7e1c1fc7021b4876bda0394462b6db4b","version_major":2,"vers
ion_minor":0}

Epoch [9/30], Training Loss: 0.4078

{"model_id":"08f9dd3a74774ea7bb38dcf26c9b8d93","version_major":2,"vers
ion_minor":0}

Epoch [9/30], Validation Accuracy: 80.66666666666667%
<torch.utils.data.dataloader.DataLoader object at 0x7e0707bb7760>

{"model_id":"65090fa97fec4646a4657e1043cfef32","version_major":2,"vers
ion_minor":0}

Epoch [10/30], Training Loss: 0.4019

{"model_id":"b5ec3389bd3249f3b955d9695781f69f","version_major":2,"vers
ion_minor":0}

Epoch [10/30], Validation Accuracy: 84.0%
<torch.utils.data.dataloader.DataLoader object at 0x7e0707bb7760>

{"model_id":"c5620dd07212451bbbe6b4cff95d4fba","version_major":2,"vers
ion_minor":0}

Epoch [11/30], Training Loss: 0.3939

{"model_id":"f1595075788e4a6d8d8e41e1681d38a4","version_major":2,"vers
ion_minor":0}

Epoch [11/30], Validation Accuracy: 80.66666666666667%
<torch.utils.data.dataloader.DataLoader object at 0x7e0707bb7760>

{"model_id":"2a1906e44a6a4ee7b31bfaa891a0aba3","version_major":2,"vers
ion_minor":0}

Epoch [12/30], Training Loss: 0.3799

{"model_id":"8185ba74d5294ccba2f5aa7f08e447d7","version_major":2,"vers
ion_minor":0}

Epoch [12/30], Validation Accuracy: 78.0%
<torch.utils.data.dataloader.DataLoader object at 0x7e0707bb7760>
```

{"model_id":"65fbe6be49614305b8b85881eeb20178","version_major":2,"version_minor":0}

```
Epoch [13/30], Training Loss: 0.3812
```

{"model_id":"70f4ba36e2374bfcb905d058d79a496c","version_major":2,"version_minor":0}

```
Epoch [13/30], Validation Accuracy: 80.0%
<torch.utils.data.dataloader.DataLoader object at 0x7e0707bb7760>
```

{"model_id":"824a3603c1564e7689ed79c0e1d81c13","version_major":2,"version_minor":0}

```
Epoch [14/30], Training Loss: 0.3776
```

{"model_id":"898781464bb348dbb65f3cfeadaba9c0","version_major":2,"version_minor":0}

```
Epoch [14/30], Validation Accuracy: 80.66666666666667%
<torch.utils.data.dataloader.DataLoader object at 0x7e0707bb7760>
```

{"model_id":"815811111f834316a351b0ea8063ce39","version_major":2,"version_minor":0}

```
Epoch [15/30], Training Loss: 0.3752
```

{"model_id":"c436e52943b249129c2563f396ea531a","version_major":2,"version_minor":0}

```
Epoch [15/30], Validation Accuracy: 74.0%
<torch.utils.data.dataloader.DataLoader object at 0x7e0707bb7760>
```

{"model_id":"ce77e07cfbc241d3a9bc5aa0c7043ffd","version_major":2,"version_minor":0}

```
Epoch [16/30], Training Loss: 0.3651
```

{"model_id":"cbb3e69c909d426996465d40f81ed23f","version_major":2,"version_minor":0}

```
Epoch [16/30], Validation Accuracy: 84.0%
<torch.utils.data.dataloader.DataLoader object at 0x7e0707bb7760>
```

{"model_id":"ce01bb298aa1459787b554f4748ee676","version_major":2,"version_minor":0}

```
Epoch [17/30], Training Loss: 0.3583
```

{"model_id":"d2a6400719b7492fa4768256b52981a8","version_major":2,"version_minor":0}

```
Epoch [17/30], Validation Accuracy: 80.66666666666667%
<torch.utils.data.dataloader.DataLoader object at 0x7e0707bb7760>

{"model_id":"67aa85b56fc04e7b898652a06e0e42be","version_major":2,"vers
ion_minor":0}

Epoch [18/30], Training Loss: 0.3597

{"model_id":"811125d04e504dcf8897ab455536dbb6","version_major":2,"vers
ion_minor":0}

Epoch [18/30], Validation Accuracy: 82.0%
<torch.utils.data.dataloader.DataLoader object at 0x7e0707bb7760>

{"model_id":"f4d32989261d49e985e6b400a36f807c","version_major":2,"vers
ion_minor":0}

Epoch [19/30], Training Loss: 0.3526

{"model_id":"c5d70718749d4a528dc038379e8a4681","version_major":2,"vers
ion_minor":0}

Epoch [19/30], Validation Accuracy: 82.66666666666667%
<torch.utils.data.dataloader.DataLoader object at 0x7e0707bb7760>

{"model_id":"f0edecb0846b4481bc6bbede9ff4d51b","version_major":2,"vers
ion_minor":0}

Epoch [20/30], Training Loss: 0.3551

{"model_id":"acd19d72e2b14be0978d1ccad7fb38e1","version_major":2,"vers
ion_minor":0}

Epoch [20/30], Validation Accuracy: 80.66666666666667%
<torch.utils.data.dataloader.DataLoader object at 0x7e0707bb7760>

{"model_id":"6c0a6f14c62a415d86d0f08002fdf2ad","version_major":2,"vers
ion_minor":0}

Epoch [21/30], Training Loss: 0.3604

{"model_id":"26646c8828a74c1db1c1ff19f4864a2c","version_major":2,"vers
ion_minor":0}

Epoch [21/30], Validation Accuracy: 79.33333333333333%
<torch.utils.data.dataloader.DataLoader object at 0x7e0707bb7760>

{"model_id":"6faf2cd947514a68b4e993587630d69d","version_major":2,"vers
ion_minor":0}

Epoch [22/30], Training Loss: 0.3422
```

{"model_id":"cf76cbd1afc9420db2a51c4bb1eb549e","version_major":2,"version_minor":0}

```
Epoch [22/30], Validation Accuracy: 86.0%
<torch.utils.data.dataloader.DataLoader object at 0x7e0707bb7760>
```

{"model_id":"d18048a3383e4b2f9e757e13ab50d5e4","version_major":2,"version_minor":0}

```
Epoch [23/30], Training Loss: 0.3321
```

{"model_id":"6685b94ae8944c488769cab1f84e9357","version_major":2,"version_minor":0}

```
Epoch [23/30], Validation Accuracy: 81.33333333333333%
<torch.utils.data.dataloader.DataLoader object at 0x7e0707bb7760>
```

{"model_id":"110df16214c244f3a153823f50c6df2d","version_major":2,"version_minor":0}

```
Epoch [24/30], Training Loss: 0.3454
```

{"model_id":"169bf49a16ea4adbb998a08b4f16ad01","version_major":2,"version_minor":0}

```
Epoch [24/30], Validation Accuracy: 83.33333333333333%
<torch.utils.data.dataloader.DataLoader object at 0x7e0707bb7760>
```

{"model_id":"74a1023e5e784d26ab8e05a81b256220","version_major":2,"version_minor":0}

```
Epoch [25/30], Training Loss: 0.3492
```

{"model_id":"271d6fd6502a4338bf43ec31ddde641c","version_major":2,"version_minor":0}

```
Epoch [25/30], Validation Accuracy: 82.66666666666667%
<torch.utils.data.dataloader.DataLoader object at 0x7e0707bb7760>
```

{"model_id":"d819f89cd1044be5a509a25b363b9e1c","version_major":2,"version_minor":0}

```
Epoch [26/30], Training Loss: 0.3465
```

{"model_id":"3f69fb10c7ba4e549cbf862eb3c616d4","version_major":2,"version_minor":0}

```
Epoch [26/30], Validation Accuracy: 82.0%
<torch.utils.data.dataloader.DataLoader object at 0x7e0707bb7760>
```

{"model_id":"8a1c2d90a3db4a3a9e1fc94dbf5ec4ce","version_major":2,"version_minor":0}

```
Epoch [27/30], Training Loss: 0.3406

{"model_id":"201840dc2a684399a585d2ebe42a1fc3","version_major":2,"vers
ion_minor":0}

Epoch [27/30], Validation Accuracy: 86.66666666666667%
<torch.utils.data.dataloader.DataLoader object at 0x7e0707bb7760>

{"model_id":"8ef570dc66f847fb85362faf8a109f85","version_major":2,"vers
ion_minor":0}

Epoch [28/30], Training Loss: 0.3225

{"model_id":"e734554f6dcc457da8c5b1874487366f","version_major":2,"vers
ion_minor":0}

Epoch [28/30], Validation Accuracy: 85.33333333333333%
<torch.utils.data.dataloader.DataLoader object at 0x7e0707bb7760>

{"model_id":"9cf216bb23d5448dbd9a0138e0a6d73e","version_major":2,"vers
ion_minor":0}

Epoch [29/30], Training Loss: 0.3319

{"model_id":"6f113943bd7a437ba0a506baba27da31","version_major":2,"vers
ion_minor":0}

Epoch [29/30], Validation Accuracy: 80.66666666666667%
<torch.utils.data.dataloader.DataLoader object at 0x7e0707bb7760>

{"model_id":"96f151f71a03467cbd5bfbec9910f33e","version_major":2,"vers
ion_minor":0}

Epoch [30/30], Training Loss: 0.3208

{"model_id":"9d59b46aef9545a4a65050d9a68a95b6","version_major":2,"vers
ion_minor":0}

Epoch [30/30], Validation Accuracy: 80.0%
```

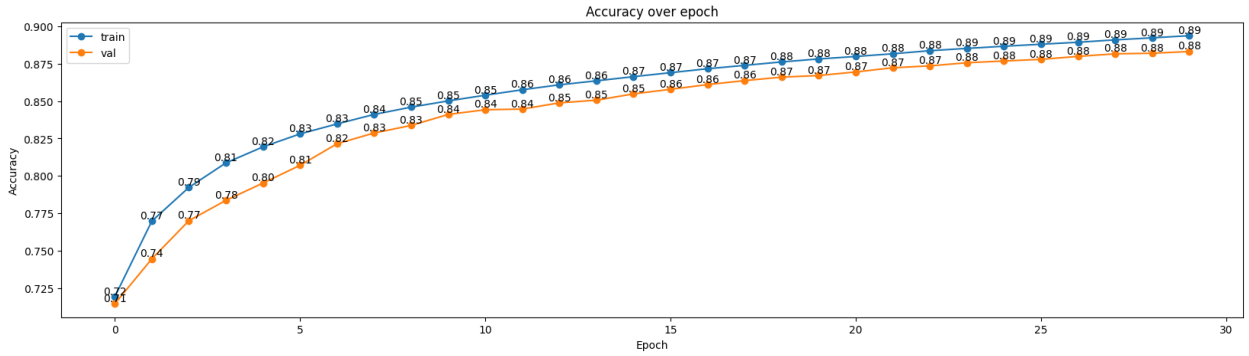
Results and Analysis

In Version 3, the accuracy for both the training and validation sets improves consistently. The discrepancy between them is maintained at just 1%, so it is safe to assume overfitting is not an issue.

However, the overall validation accuracy did not improve from the previous version. To potentially enhance model performance, a more complex CNN architecture may be needed.

```
print(f"Third version of model")
plt.figure(figsize=(20,5))
plt.plot(train_auc_epoch, '-o', label="train")
plt.plot(val_auc_epoch, '-o', label="val")
# Adding value labels
for i, value in enumerate(train_auc_epoch):
    plt.text(i, value, f"{value:.2f}", ha='center', va='bottom')

for i, value in enumerate(val_auc_epoch):
    plt.text(i, value, f"{value:.2f}", ha='center', va='bottom')
plt.xlabel("Epoch")
plt.ylabel("Accuracy")
plt.title("Accuracy over epoch")
plt.legend()
```

```
Third version of model

<matplotlib.legend.Legend at 0x7e07077f47c0>
```




```
import pickle
# Save the lists to a file
with open('model_metrics3.pkl', 'wb') as f:
    pickle.dump([train_auc, val_auc, train_auc_epoch, val_auc_epoch],
f)

import pickle
with open('model_metrics3.pkl', 'rb') as f:
    train_auc, val_auc, train_auc_epoch, val_auc_epoch =
pickle.load(f)
```

Conclusion

The overall performance of the model is satisfactory, with accuracy reaching the 90% level. Overfitting was an issue in Version 1, but using a more complex CNN architecture with regularization layers has significantly reduced overfitting.

The accuracy has improved gradually and consistently, albeit slowly, in Version 3. While continuing to train for more epochs might yield slight improvements, it appears that the current network architecture has already captured most of the data patterns it can.

To further improve the model, I would try additional data augmentation and possibly add more CNN layers and fully connected layers. This could help the model capture more subtle and complex patterns that were previously missed.

```
model.load_state_dict(torch.load('best_model.pt'))
```

```
<All keys matched successfully>
```

```
model.eval()

predictions = []

for i, (images, labels) in enumerate(tqdm(test_dataloader,
total=int(len(test_dataloader)))):
    images = images.to(device)
    labels = labels.to(device)

    outputs = model(images)
    pred = outputs[:,1].detach().cpu().numpy()

    for j in pred:
        predictions.append(j)
```

```
{"model_id":"c7e68d224deb4a0a800eb5713c6001b0","version_major":2,"vers
ion_minor":0}
```

```
dir_main =
'/content/drive/MyDrive/CuBoulder/DTSA5511DeepLearning/wk3/CancerData'
#path = dir_main + '/train/'
submission_file = dir_main + '/sample_submission.csv'
submission_df = pd.read_csv(submission_file)

sub['label'] = predictions
sub.loc[sub['label'] >= 0.5, 'label_zero_one'] = 1
sub.loc[sub['label'] < 0.5, 'label_zero_one'] = 0

# Merge the two DataFrames based on the 'id' column
merged_df = pd.merge(submission_df, sub, on='id', how='left')
# Update the 'label' column in the merged DataFrame
merged_df['label'] = merged_df['label_zero_one'].fillna(0.0)
merged_df['label'] = merged_df['label'].astype(int)
# Save the updated DataFrame back to the original submission_df
submission_df = merged_df[['id', 'label']]
submission_df.to_csv('submission.csv', index=False)
```

```python
# save submission.csv to local storage
from google.colab import files
files.download('submission.csv')
```

```
<IPython.core.display.Javascript object>

<IPython.core.display.Javascript object>
```

```python
test_images = np.random.choice(sub.id, size=50, replace=False)
#test_images = np.random.choice(sub.id, size=10, replace=False) #my
change from 50 to 10

fig, ax = plt.subplots(5, 10, figsize=(20,10))

for n in range(5):
    for m in range(10):
        img_id = test_images[m + n*10]
        image = plt.imread(test_path + img_id + ".tif")
        pred = sub.loc[sub['id'] == img_id,
'label_zero_one'].values[0]
        label = "Cancer" if(pred >= 0.5) else "Healthy"
        ax[n,m].imshow(image)
        ax[n,m].grid(False)
        ax[n,m].tick_params(labelbottom=False, labelleft=False)
        ax[n,m].set_title("Label: " + label)
```