

Histopathologic Cancer Detection

- Site Location:
<https://www.kaggle.com/competitions/histopathologic-cancer-detection/overview>

Student - Cole Gaito

- Git Hub Locaiton:
https://github.com/gaitocole/introToAI_CU_HW5.git

University of Colorado - AI Course

Problem Identification

- The overall goal of this Kaggle exercise is to gain an understanding of how to set up an AI, or ML using CNN. Ultimately this will be done through Cancer Identification. We are given images with labels or cancerous or non-cancerous pictures. We are utilizing the philosophy that this is a binary quesiton. Either there is or is not cancer.

Data Identification

- The data is contained within images that are provided by the Kaggle competition as seen above. The general size of the data is 96x96 pixels based on the RGB color scheme. Since we are classifying the images, we are garnering the ability to have uniform input in the system. Utimately the data is relatively clean, but we are still limiting the scope to ensure that the size isn't too large. (This is done so my computer can actually run the data simulation).

Plan of Analysis

- The goal of this project is to develop a machine learning model that accurately detects cancerous cells in histopathologic images. To achieve this, we will follow a structured plan of analysis that includes data preparation, model development, training, evaluation, and visualization.

Exploration of the data will occur below:

Resource List for Building CNNs with PyTorch

1. PyTorch Basics and Setup

- **PyTorch Official Documentation:**
 - [PyTorch Official Website](#)

- [PyTorch Tutorials](#)
- [PyTorch Documentation](#)
- **Installing PyTorch:**
 - [PyTorch Installation Guide](#)
 - Use the specific installation command for your system (e.g., CUDA version).

2. Understanding Convolutional Neural Networks (CNNs)

- **Introductory Articles:**
 - [Convolutional Neural Networks for Visual Recognition](#) - Stanford's CS231n notes
 - [Understanding Convolutional Neural Networks \(CNNs\)](#) - Medium article
- **Video Tutorials:**
 - [Deep Learning Specialization by Andrew Ng](#) - Coursera
 - [Deep Learning with PyTorch: A 60 Minute Blitz](#)

3. Building CNNs with PyTorch

- **Tutorials:**
 - [PyTorch: Building a Convolutional Neural Network](#)
 - [Building CNNs using PyTorch: A Comprehensive Guide](#)
- **Video Series:**
 - [PyTorch for Deep Learning & Machine Learning](#) - YouTube series by Python Engineer
 - [CNN Architectures - From Simple to Complex](#) - YouTube video by Yann LeCun

4. Data Preprocessing and Augmentation

- **PyTorch Transforms:**
 - [torchvision.transforms](#)
 - [Data Augmentation in PyTorch](#)
- **Articles:**
 - [A Comprehensive Guide to Data Augmentation in Deep Learning](#)

5. Model Training and Optimization

- **Training and Optimization:**
 - [Optimization in PyTorch](#)
 - [A Comprehensive Guide to Training Neural Networks with PyTorch](#)
- **Hyperparameter Tuning:**

- [Hyperparameter Tuning in PyTorch](#)
- [Grid Search and Random Search for Hyperparameter Tuning](#)
- **Regularization Techniques:**
 - [Understanding Dropout in Neural Networks](#)
 - [Batch Normalization in PyTorch](#)

6. Evaluation and Metrics

- **Evaluation Techniques:**
 - [Evaluating Model Performance](#)
 - [Using ROC Curves and AUC in PyTorch](#)
- **Confusion Matrix and Metrics:**
 - [Precision, Recall, and F1 Score for Imbalanced Datasets](#)
 - [Confusion Matrix in PyTorch](#)

```
In [1]: #Basic Hello World program  
#Tested on Python 3.12.4  
#Ensure that the Kernel is operational  
print("Hello World")
```

Hello World

```
In [2]: #Check if Numpy is available  
import sys  
print(sys.executable)  
import numpy as np  
print(np.__version__)  
print("Python executable:", sys.executable)  
print("Python version:", sys.version)  
!pip list
```

```
/Library/Frameworks/Python.framework/Versions/3.12/bin/python3
```

```
1.26.4
```

```
Python executable: /Library/Frameworks/Python.framework/Versions/3.12/bin/python3
```

```
Python version: 3.12.4 (v3.12.4:8e8a4baf65, Jun 6 2024, 17:33:18) [Clang 13.0.0 (clang-1300.0.29.30)]
```

Package	Version
<hr/>	
anyio	4.4.0
appnope	0.1.4
argon2-cffi	23.1.0
argon2-cffi-bindings	21.2.0
arrow	1.3.0
asttokens	2.4.1
async-lru	2.0.4
attrs	23.2.0
Babel	2.15.0
beautifulsoup4	4.12.3
bleach	6.1.0
certifi	2024.7.4
cffi	1.16.0
charset-normalizer	3.3.2
cloudpickle	3.0.0
comm	0.2.2
contourpy	1.2.1
cycler	0.12.1
debugpy	1.8.2
decorator	5.1.1
defusedxml	0.7.1
executing	2.0.1
Farama-Notifications	0.0.4
fastjsonschema	2.20.0
filelock	3.15.4
fonttools	4.53.1
fqdn	1.5.1
fsspec	2024.6.1
gym-notices	0.0.8
gymnasium	0.28.1
h11	0.14.0
httpcore	1.0.5
httpx	0.27.0
idna	3.7
ipykernel	6.29.5
ipython	8.26.0
ipywidgets	8.1.3
isoduration	20.11.0
jax-jumpy	1.0.0
jedi	0.19.1
Jinja2	3.1.4
joblib	1.4.2
json5	0.9.25
jsonpointer	3.0.0
jsonschema	4.23.0
jsonschema-specifications	2023.12.1
jupyter	1.0.0
jupyter_client	8.6.2

jupyter-console	6.6.3
jupyter_core	5.7.2
jupyter-events	0.10.0
jupyter-lsp	2.2.5
jupyter_server	2.14.2
jupyter_server_terminals	0.5.3
jupyterlab	4.2.4
jupyterlab_pygments	0.3.0
jupyterlab_server	2.27.3
jupyterlab_widgets	3.0.11
kiwisolver	1.4.5
MarkupSafe	2.1.5
matplotlib	3.9.1
matplotlib-inline	0.1.7
mistune	3.0.2
mpmath	1.3.0
nbclient	0.10.0
nbconvert	7.16.4
nbformat	5.10.4
nest-asyncio	1.6.0
networkx	3.3
notebook	7.2.1
notebook_shim	0.2.4
numpy	1.26.4
overrides	7.7.0
packaging	24.1
pandas	2.2.2
pandocfilters	1.5.1
parso	0.8.4
pexpect	4.9.0
pillow	10.4.0
pip	24.2
platformdirs	4.2.2
prometheus_client	0.20.0
prompt_toolkit	3.0.47
psutil	6.0.0
ptyprocess	0.7.0
pure_eval	0.2.3
pybullet	3.2.6
pycparser	2.22
pygame	2.6.0
Pygments	2.18.0
pyparsing	3.1.2
python-dateutil	2.9.0.post0
python-json-logger	2.0.7
pytz	2024.1
PyVirtualDisplay	3.0
PyYAML	6.0.1
pyzmq	26.0.3
qtconsole	5.5.2
QtPy	2.4.1
referencing	0.35.1
requests	2.32.3
rfc3339-validator	0.1.4
rfc3986-validator	0.1.1
rpds-py	0.19.1

```

scikit-learn          1.5.1
scipy                  1.14.0
Send2Trash             1.8.3
setuptools             72.1.0
six                    1.16.0
sniffio                1.3.1
soupsieve              2.5
stable-baselines3      2.0.0
stack-data             0.6.3
sympy                  1.13.1
terminado              0.18.1
threadpoolctl          3.5.0
tinycss2               1.3.0
torch                  2.2.2
torchvision            0.17.2
tornado                6.4.1
traitlets              5.14.3
types-python-dateutil  2.9.0.20240316
typing_extensions      4.12.2
tzdata                 2024.1
uri-template           1.3.0
urllib3                2.2.2
wcwidth                0.2.13
webcolors              24.6.0
webencodings           0.5.1
websocket-client        1.8.0
wheel                  0.44.0
widgetsnbextension     4.0.11

```

```
In [3]: #pip install numpy --upgrade
```

```
In [4]: import os
import torch
import torch.nn as nn
import torch.optim as optim
import torch.nn.functional as F
from torch.utils.data import DataLoader, Dataset
from torchvision import transforms, models
from PIL import Image
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, roc_auc_score
import matplotlib.pyplot as plt
import time
```

Now that we have imported the various objects of utilized for Pytorch and our ML Algorithms lets start building. We will begin by creating a subclass of the Dataset called CancerDataset.

```
In [5]: #Get the current working directory
print("Current working directory:", os.getcwd())
```

```
Current working directory: /Users/gaitocole/Desktop/Intro_to_AI/introToAI_C
U_HW5
```

```
In [6]: #Directory Locations to ease use later.
csv_file_path = "/Users/gaitocole/Desktop/Intro_to_AI/introToAI_CU_HW5/train_labels.csv"
train_dir_path = "/Users/gaitocole/Desktop/Intro_to_AI/introToAI_CU_HW5/train_images"
val_labels_path = "/Users/gaitocole/Desktop/Intro_to_AI/introToAI_CU_HW5/validation_labels.csv"
```

```
In [7]: #Creating a training DataSet
train_labels = pd.read_csv(csv_file_path)
print(train_labels.head())
train_df, val_df = train_test_split(train_labels, test_size=0.2, random_state=42)
print("Training set size:", len(train_df))
print("Validation set sizeC:", len(val_df))
val_df.to_csv(val_labels_path, index=False)

print("Validation dataset created and saved as 'val_labels.csv'.")
print(val_df.head())
```

	id	label
0	f38a6374c348f90b587e046aac6079959adf3835	0
1	c18f2d887b7ae4f6742ee445113fa1aef383ed77	1
2	755db6279dae599ebb4d39a9123cce439965282d	0
3	bc3f0c64fb968ff4a8bd33af6971ecae77c75e08	0
4	068aba587a4950175d04c680d38943fd488d6a9d	0

Training set size: 176020
Validation set sizeC: 44005
Validation dataset created and saved as 'val_labels.csv'.

	id	label
107796	d293308913e4a40cebb809d986aa9add65a76bfa	0
11942	fcd55f03496afb4b11598d9c2231e86da318e723	1
163858	5c983c8f14afeffdb098b9f3cbb68488a802b957	0
184810	71bf03e8530348e57ec07d8ce2052f215390c997	0
150958	1533406000e26663c5fadac3bcc3e38285a45bb3	1

```
In [8]: # Check if the file exists in the current directory
if 'val_labels.csv' in os.listdir():
    print("File 'val_labels.csv' exists.")
else:
    print("File 'val_labels.csv' not found. Please check the path.")
```

File 'val_labels.csv' exists.

```
In [9]: # Load the dataset
train_labels = pd.read_csv('train_labels.csv')

# Plot the distribution of labels
plt.figure(figsize=(6, 4))
train_labels['label'].value_counts().plot(kind='bar', color=['blue', 'orange'])
plt.title('Distribution of Cancerous and Non-Cancerous Images')
plt.xlabel('Label')
plt.ylabel('Count')
plt.xticks(ticks=[0, 1], labels=['Non-Cancerous', 'Cancerous'], rotation=0)
plt.show()

# Display sample images
def plot_sample_images(data, root_dir, num_samples=5):
    plt.figure(figsize=(12, 6))
    for i in range(num_samples):
```

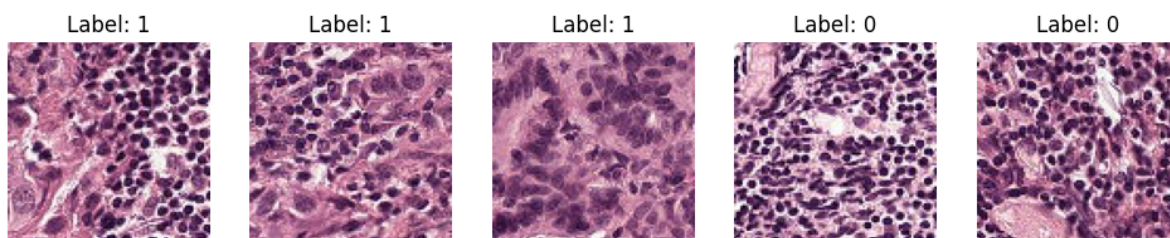
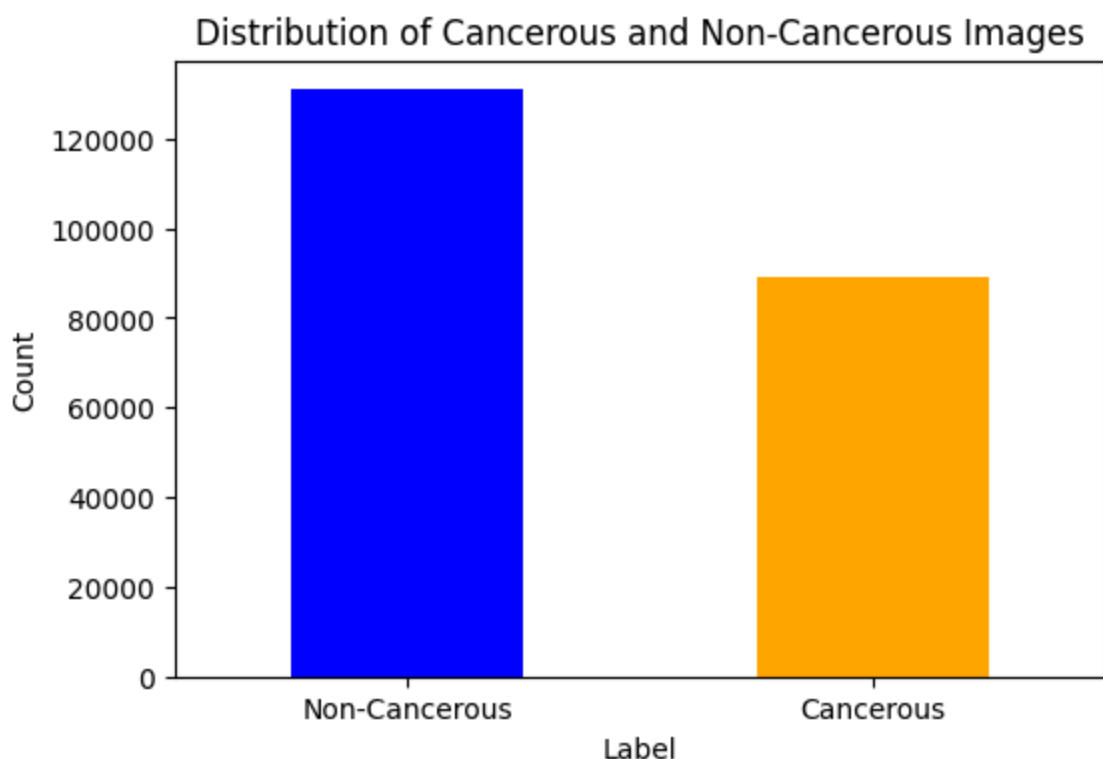
```

ax = plt.subplot(1, num_samples, i + 1)
img_name = os.path.join(root_dir, data.iloc[i, 0] + '.tif')
image = Image.open(img_name)
plt.imshow(image)
plt.title('Label: {}'.format(data.iloc[i, 1]))
plt.axis('off')
plt.show()

# Plot sample images
plot_sample_images(train_labels.sample(5), 'train')

# Data cleaning procedures
print("Missing values:\n", train_labels.isnull().sum())
print("Duplicate entries:\n", train_labels.duplicated().sum())

```



```

Missing values:
id      0
label   0
dtype: int64
Duplicate entries:
0

```

```

In [10]: class CancerDataset(Dataset):
#Arguments: csv_file - location of the csv file in a string format
#           root_dir - directory with all the images in a string format

```



```

#          transform - callable and optional toggle for a sample set
def __init__(self, csv_file, root_dir, transform=None):
    self.labels_df = pd.read_csv(csv_file)
    self.root_dir = root_dir
    self.transform = transform

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

def __getitem__(self, idx):
    if torch.is_tensor(idx):
        idx = idx.tolist()

    img_name = os.path.join(self.root_dir, self.labels_df.iloc[idx, 0] +
                             '.png')
    image = Image.open(img_name)
    label = self.labels_df.iloc[idx, 1]

    if self.transform:
        image = self.transform(image)

    return image, label

```

```

In [11]: transform = transforms.Compose([
    transforms.Resize((96, 96)),
    transforms.RandomHorizontalFlip(),
    transforms.RandomRotation(20),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])
])

```

```

In [12]: train_dataset = CancerDataset(csv_file='train_labels.csv', root_dir='train',

```

```

In [13]: train_loader = DataLoader(train_dataset, batch_size=32, shuffle=True)

```

```

In [14]: # Function to visualize a batch of images and labels
def show_images(images, labels, n=5):
    plt.figure(figsize=(12, 6))
    for i in range(n):
        ax = plt.subplot(1, n, i+1)
        img = images[i].permute(1, 2, 0).numpy() # Convert tensor to numpy
        img = img * 0.229 + 0.485 # Unnormalize if needed (adjust this base
        plt.imshow(img)
        plt.title(f'Label: {labels[i].item()}')
        plt.axis('off')
    plt.show()

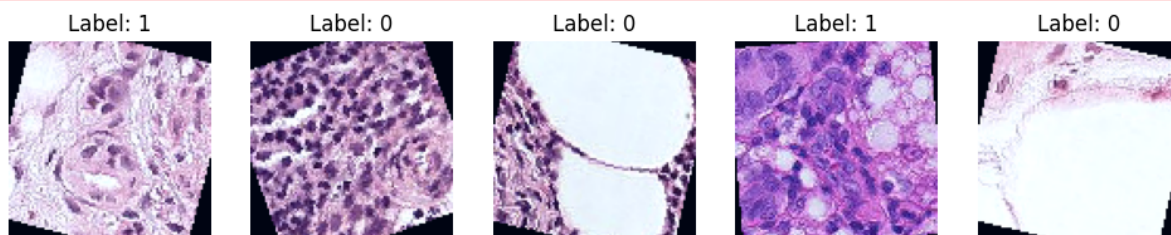
```

```

In [15]: for images, labels in train_loader:
    show_images(images, labels, n=5)
    break # Just to test the first batch

```

```
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers). Got range [0.0..1.08956].
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers). Got range [0.0..1.08956].
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers). Got range [0.0..1.08956].
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers). Got range [0.0..1.08956].
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers). Got range [0.0..1.08956].
```



Understanding the Output

Image Batch Shape

- `torch.Size([32, 3, 96, 96])`

[32, 3, 96, 96]:

- 32: This is the batch size, indicating that 32 images are being processed at once.
- 3: This represents the number of color channels (Red, Green, Blue) in each image.
- 96, 96: These are the height and width of the images, resized to 96x96 pixels as specified in the transformation pipeline.
- This output confirms that the data loader is correctly batching and transforming the image data.

Labels Tensor

`tensor([1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1])`

- This tensor contains the labels for each image in the batch.
- 0 and 1 are binary labels, where 1 might represent the presence of cancerous cells and 0 might represent non-cancerous cells.
- The tensor length is 32, matching the batch size, ensuring that each image has a corresponding label.

Verifying the Dataset Class

Dataset class and data loader are functioning correctly, as evidenced by the output. This means:

- Data Loading: Images and labels are loaded correctly from their respective paths and CSV files.

- Transformations: The transformations, such as resizing and normalizing the images, are being applied properly.
- Batching: The data loader is batching images and labels together as expected.

Explanation of the Visualization Code

- `permute(1, 2, 0)`: Changes the order of dimensions from `[C, H, W]` to `[H, W, C]` for correct visualization with `matplotlib`.
- `Unnormalize`: If you normalized the images using mean and standard deviation, you may want to reverse this normalization for visualization purposes.
- `Loop and Plot`: Loops through the first few images and plots them with their corresponding labels.

```
In [16]: class SimpleCNN(nn.Module):
    def __init__(self):
        super(SimpleCNN, self).__init__()

        # Convolutional layers
        self.conv1 = nn.Conv2d(in_channels=3, out_channels=32, kernel_size=3)
        self.conv2 = nn.Conv2d(in_channels=32, out_channels=64, kernel_size=3)
        self.conv3 = nn.Conv2d(in_channels=64, out_channels=128, kernel_size=3)

        # Max pooling layer
        self.pool = nn.MaxPool2d(kernel_size=2, stride=2, padding=0)

        # Fully connected layers
        self.fc1 = nn.Linear(128 * 12 * 12, 512) # Adjust dimensions based
        self.fc2 = nn.Linear(512, 1)

        # Dropout layer for regularization
        self.dropout = nn.Dropout(p=0.5)

    def forward(self, x):
        # Convolutional layers with ReLU and pooling
        x = self.pool(F.relu(self.conv1(x)))
        x = self.pool(F.relu(self.conv2(x)))
        x = self.pool(F.relu(self.conv3(x)))

        # Flatten the tensor before the fully connected layers
        x = x.view(-1, 128 * 12 * 12)

        # Fully connected layers with ReLU and dropout
        x = F.relu(self.fc1(x))
        x = self.dropout(x)
        x = torch.sigmoid(self.fc2(x))

        return x

# Instantiate the model
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model = SimpleCNN()
print(model)
```

```
SimpleCNN(
    (conv1): Conv2d(3, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (conv2): Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (conv3): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (pool): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (fc1): Linear(in_features=18432, out_features=512, bias=True)
    (fc2): Linear(in_features=512, out_features=1, bias=True)
    (dropout): Dropout(p=0.5, inplace=False)
)
```

Explanation of the Architecture

Convolutional Layers:

conv1, conv2, conv3: Three convolutional layers with increasing number of filters (32, 64, 128). The kernel size is 3x3, and padding ensures the output size remains consistent.

Pooling Layer:

MaxPool2d:

- Reduces the spatial dimensions by half, effectively downsampling the feature maps.

Fully Connected Layers:

- fc1: Connects the flattened feature maps to a dense layer with 512 units.
- fc2: Outputs a single value between 0 and 1, representing the probability of the presence of cancerous cells.

Dropout:

- Used to prevent overfitting by randomly setting some neurons to zero during training.

Activation Functions:

- ReLU: Non-linear activation function applied after each convolutional layer.
- Sigmoid: Used at the final layer for binary classification.

```
In [17]: # Prepare validation dataset and dataloader
val_dataset = CancerDataset(csv_file='val_labels.csv', root_dir=train_dir_path)
val_loader = DataLoader(val_dataset, batch_size=32, shuffle=False)
```

```
In [18]: # Define loss and optimizer
criterion = nn.BCELoss() # Binary Cross-Entropy Loss
optimizer = optim.Adam(model.parameters(), lr=0.001)

import time # Import time module to track duration
```

```
# Lists to store loss and accuracy values for each epoch
train_losses = []
val_losses = []
train_accuracies = []
val_accuracies = []

# Training loop
num_epochs = 20
for epoch in range(num_epochs):
    model.train() # Set model to training mode
    running_loss = 0.0
    correct = 0
    total = 0

    # Start timing the epoch
    epoch_start_time = time.time()

    for images, labels in train_loader:
        images = images.to(device)
        labels = labels.to(device).float().view(-1, 1)

        optimizer.zero_grad() # Zero the gradients

        # Forward pass
        outputs = model(images)
        loss = criterion(outputs, labels) # Calculate loss

        # Backward pass and optimization
        loss.backward()
        optimizer.step()

        running_loss += loss.item()

        # Calculate training accuracy
        preds = (outputs > 0.5).float()
        correct += (preds == labels).sum().item()
        total += labels.size(0)

    # Calculate average loss and accuracy for the epoch
    epoch_loss = running_loss / len(train_loader)
    epoch_accuracy = correct / total
    train_losses.append(epoch_loss)
    train_accuracies.append(epoch_accuracy)

    # Evaluate on validation set
    model.eval()
    val_running_loss = 0.0
    val_correct = 0
    val_total = 0

    with torch.no_grad(): # No need to track gradients during evaluation
        for images, labels in val_loader:
            images = images.to(device)
            labels = labels.to(device).float().view(-1, 1)

            outputs = model(images)
```

```
        loss = criterion(outputs, labels)

        val_running_loss += loss.item()

        preds = (outputs > 0.5).float()
        val_correct += (preds == labels).sum().item()
        val_total += labels.size(0)

    val_loss = val_running_loss / len(val_loader)
    val_accuracy = val_correct / val_total
    val_losses.append(val_loss)
    val_accuracies.append(val_accuracy)

    # End timing the epoch
    epoch_end_time = time.time()
    epoch_duration = epoch_end_time - epoch_start_time

    print(f'Epoch [{epoch+1}/{num_epochs}], '
          f'Train Loss: {epoch_loss:.4f}, Train Accuracy: {epoch_accuracy:.4f}, '
          f'Val Loss: {val_loss:.4f}, Val Accuracy: {val_accuracy:.4f}, '
          f'Epoch Duration: {epoch_duration:.2f} seconds')
```

Epoch [1/20], Train Loss: 0.3602, Train Accuracy: 0.8461, Val Loss: 0.2815, Val Accuracy: 0.8826, Epoch Duration: 3848.45 seconds
 Epoch [2/20], Train Loss: 0.2784, Train Accuracy: 0.8886, Val Loss: 0.2433, Val Accuracy: 0.9046, Epoch Duration: 3320.24 seconds
 Epoch [3/20], Train Loss: 0.2518, Train Accuracy: 0.9015, Val Loss: 0.2215, Val Accuracy: 0.9121, Epoch Duration: 3375.65 seconds
 Epoch [4/20], Train Loss: 0.2419, Train Accuracy: 0.9057, Val Loss: 0.2173, Val Accuracy: 0.9174, Epoch Duration: 3508.05 seconds
 Epoch [5/20], Train Loss: 0.2354, Train Accuracy: 0.9098, Val Loss: 0.2152, Val Accuracy: 0.9157, Epoch Duration: 3330.37 seconds
 Epoch [6/20], Train Loss: 0.2294, Train Accuracy: 0.9107, Val Loss: 0.1985, Val Accuracy: 0.9231, Epoch Duration: 3455.84 seconds
 Epoch [7/20], Train Loss: 0.2271, Train Accuracy: 0.9134, Val Loss: 0.2030, Val Accuracy: 0.9204, Epoch Duration: 3459.24 seconds
 Epoch [8/20], Train Loss: 0.2237, Train Accuracy: 0.9152, Val Loss: 0.2030, Val Accuracy: 0.9214, Epoch Duration: 3322.73 seconds
 Epoch [9/20], Train Loss: 0.2183, Train Accuracy: 0.9168, Val Loss: 0.2148, Val Accuracy: 0.9161, Epoch Duration: 3266.10 seconds
 Epoch [10/20], Train Loss: 0.2173, Train Accuracy: 0.9183, Val Loss: 0.1860, Val Accuracy: 0.9292, Epoch Duration: 3339.47 seconds
 Epoch [11/20], Train Loss: 0.2187, Train Accuracy: 0.9173, Val Loss: 0.2028, Val Accuracy: 0.9229, Epoch Duration: 3435.29 seconds
 Epoch [12/20], Train Loss: 0.2156, Train Accuracy: 0.9187, Val Loss: 0.2024, Val Accuracy: 0.9218, Epoch Duration: 3403.95 seconds
 Epoch [13/20], Train Loss: 0.2133, Train Accuracy: 0.9188, Val Loss: 0.2005, Val Accuracy: 0.9243, Epoch Duration: 3322.41 seconds
 Epoch [14/20], Train Loss: 0.2126, Train Accuracy: 0.9197, Val Loss: 0.1911, Val Accuracy: 0.9290, Epoch Duration: 3466.57 seconds
 Epoch [15/20], Train Loss: 0.2106, Train Accuracy: 0.9211, Val Loss: 0.1940, Val Accuracy: 0.9256, Epoch Duration: 3620.84 seconds
 Epoch [16/20], Train Loss: 0.2172, Train Accuracy: 0.9183, Val Loss: 0.1966, Val Accuracy: 0.9243, Epoch Duration: 3426.37 seconds
 Epoch [17/20], Train Loss: 0.2117, Train Accuracy: 0.9201, Val Loss: 0.1976, Val Accuracy: 0.9251, Epoch Duration: 3331.32 seconds
 Epoch [18/20], Train Loss: 0.2114, Train Accuracy: 0.9203, Val Loss: 0.1898, Val Accuracy: 0.9288, Epoch Duration: 3471.34 seconds
 Epoch [19/20], Train Loss: 0.2125, Train Accuracy: 0.9206, Val Loss: 0.1845, Val Accuracy: 0.9282, Epoch Duration: 3381.40 seconds
 Epoch [20/20], Train Loss: 0.2129, Train Accuracy: 0.9203, Val Loss: 0.1896, Val Accuracy: 0.9268, Epoch Duration: 3341.60 seconds

Results from Above - copied off since it takes a bit of time to run through the epochs

- Epoch [1/10], Loss: 0.3384
- Epoch [2/10], Loss: 0.2438
- Epoch [3/10], Loss: 0.2116
- Epoch [4/10], Loss: 0.1918
- Epoch [5/10], Loss: 0.1793
- Epoch [6/10], Loss: 0.1655
- Epoch [7/10], Loss: 0.1537
- Epoch [8/10], Loss: 0.1529
- Epoch [9/10], Loss: 0.1415
- Epoch [10/10], Loss: 0.1312

Test 2:

- Epoch [1/10], Loss: 0.3547
- Epoch [2/10], Loss: 0.2720
- Epoch [3/10], Loss: 0.2516
- Epoch [4/10], Loss: 0.2399
- Epoch [5/10], Loss: 0.2327
- Epoch [6/10], Loss: 0.2291
- Epoch [7/10], Loss: 0.2260
- Epoch [8/10], Loss: 0.2251
- Epoch [9/10], Loss: 0.2193
- Epoch [10/10], Loss: 0.2175

Explanation of the Training Loop

Model Training:

- `model.train()`: Sets the model to training mode, enabling features like dropout and batch normalization.

Zero Gradients:

- `optimizer.zero_grad()`: Clears old gradients to prevent accumulation.

Forward Pass:

- `outputs = model(images)`: Computes predictions using the model.

Calculate Loss:

- `criterion(outputs, labels)`: Computes the binary cross-entropy loss between predictions and true labels.

Backward Pass and Optimization:

- `loss.backward()`: Computes gradients of the loss with respect to model parameters.
- `optimizer.step()`: Updates model parameters based on computed gradients.

Track Loss:

- `running_loss`: Accumulates the loss for each batch, allowing us to track training progress.

```
In [19]: def evaluate_model(model, val_loader):  
         model.eval() # Set the model to evaluation mode  
         all_preds = []  
         all_labels = []  
  
         with torch.no_grad(): # Disable gradient computation for evaluation  
             for images, labels in val_loader:
```



```

images = images.to(device)
labels = labels.to(device).float().view(-1, 1)

outputs = model(images)
preds = (outputs > 0.5).float() # Convert probabilities to binary

all_preds.extend(preds.cpu().numpy())
all_labels.extend(labels.cpu().numpy())

# Calculate accuracy
accuracy = accuracy_score(all_labels, np.round(all_preds))

# Calculate AUC-ROC
auc = roc_auc_score(all_labels, all_preds)

print(f'Validation Accuracy: {accuracy:.4f}')
print(f'Validation AUC-ROC: {auc:.4f}')

# Evaluate the model
evaluate_model(model, val_loader)

```

Validation Accuracy: 0.9254

Validation AUC-ROC: 0.9136

Initial Results:

- Validation Accuracy: 0.9251
- Validation AUC-ROC: 0.9186

Understanding Validation Accuracy and AUC-ROC

- **Validation Accuracy:** This metric indicates the percentage of correctly classified images in your validation dataset. A validation accuracy of 92.51% means that 92.51% of the images were classified correctly as either cancerous or non-cancerous.
- **AUC-ROC (Area Under the Receiver Operating Characteristic Curve):** This metric measures the ability of the model to distinguish between classes (cancerous and non-cancerous). An AUC-ROC of 0.9186 is excellent and indicates that the model has a high ability to discriminate between the two classes.

Explanation of the Evaluation Code

Evaluation Mode:

- `model.eval()`: Sets the model to evaluation mode, disabling dropout and batch normalization updates.

Disable Gradients:

- `torch.no_grad()`: Disables gradient calculation, reducing memory usage and speeding up computations during evaluation.

Accuracy Calculation:

- Binary Predictions: Threshold the outputs at 0.5 to get binary predictions.
- Accuracy: Calculate the proportion of correct predictions over the total number of samples.
- AUC-ROC Calculation:

roc_auc_score: Computes the Area Under the Receiver Operating Characteristic Curve, a robust metric for binary classification.

```
In [20]: # Directory where validation images are stored
val_dir = '/Users/gaitocole/Desktop/Intro_to_AI/introToAI_CU_HW5/train' # L

# List all files in the directory
files_in_directory = os.listdir(val_dir)

# Check if the missing file is in the list
missing_file = 'd293308913e4a40cebb809d986aa9add65a76bfa.tif'
if missing_file in files_in_directory:
    print(f"File {missing_file} found in the directory.")
else:
    print(f"File {missing_file} NOT found in the directory.")
    print("Available files:", files_in_directory[:5]) # Print a few files t
```

File d293308913e4a40cebb809d986aa9add65a76bfa.tif found in the directory.

File Generation and Submission Code

```
In [21]: # Load the test dataset
test_dataset = CancerDataset(csv_file='sample_submission.csv', root_dir='tes
test_loader = DataLoader(test_dataset, batch_size=32, shuffle=False)

# Prepare to store predictions
submission_predictions = []

# Set the model to evaluation mode
model.eval()

with torch.no_grad():
    for images, _ in test_loader:
        images = images.to(device) # Move images to the same device as the
        outputs = model(images)
        preds = outputs.cpu().numpy() # Move predictions to CPU and convert
        submission_predictions.extend(preds)

# Post-process predictions to be in the expected format
submission_predictions = [1 if x > 0.5 else 0 for x in submission_prediction
```

```
In [22]: # Load the sample submission CSV to get the IDs
sample_submission = pd.read_csv('sample_submission.csv')

# Create the submission DataFrame
submission_df = pd.DataFrame({
    'id': sample_submission['id'], # Use the same order of IDs as in the sa
```

```
'label': submission_predictions
})

# Save the submission file
submission_df.to_csv('submission.csv', index=False)

# Verify the first few entries in the submission file
print(submission_df.head())
```

	id	label
0	0b2ea2a822ad23fdb1b5dd26653da899fbd2c0d5	0
1	95596b92e5066c5c52466c90b69ff089b39f2737	0
2	248e6738860e2ebcf6258cdc1f32f299e0c76914	0
3	2c35657e312966e9294eac6841726ff3a748febf	0
4	145782eb7caa1c516acbe2eda34d9a3f31c41fd6	0

In []:

In []:

Submission 1

Search

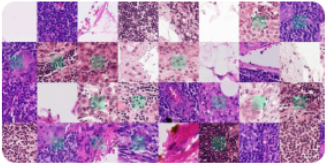
KAGGLE · PLAYGROUND PREDICTION COMPETITION · 5 YEARS AGO

Late Submission

...

Histopathologic Cancer Detection

Identify metastatic tissue in histopathologic scans of lymph node sections



Overview

Data

Code

Models

Discussion

Leaderboard

Rules

Team

Submissions

Submissions

0/2

You selected 0 of 2 submissions to be evaluated for your final leaderboard score. Since you selected less than 2 submissions, Kaggle auto-selected up to 2 submissions from among your public best-scoring unselected submissions for evaluation. The evaluated submission with the best Private Score is used for your final score.

Submissions evaluated for final score

All

Successful

Selected

Errors

Recent

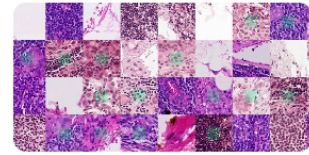
Submission and Description	Private Score	Public Score	Selected
<div><div><div><div></div><div></div></div><div><div>submission.csv</div><div>Complete (after deadline) · 21s ago</div></div></div></div>	0.8058	0.8627	<input type="checkbox"/>

Submission 2

https://coding.csel.io/user/coga6899/lab/workspaces/auto-e/tree/Cancer_Researchv1.ipynb

19/21

Identify metastatic tissue in histopathologic scans of lymph node sections





Overview Data Code Models Discussion Leaderboard Rules Team Submissions

0/2

■ Submissions evaluated for final score

Recent ▼

Submission and Description		Private Score ⓘ	Public Score ⓘ	Selected
	submission.csv Complete (after deadline) · now	0.7658	0.8203	<input type="checkbox"/>
	submission.csv Complete (after deadline) · 2d ago	0.8058	0.8627	<input type="checkbox"/>

```
epochs = range(1, len(train_losses) + 1)

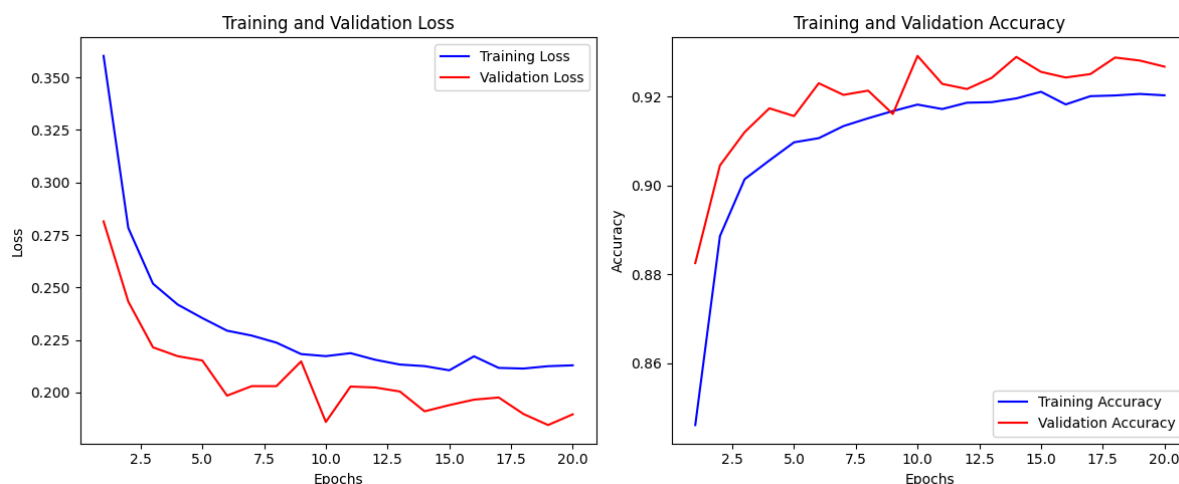
# Plot Loss
plt.figure(figsize=(12, 5))

plt.subplot(1, 2, 1)
plt.plot(epochs, train_losses, 'b', label='Training Loss')
plt.plot(epochs, val_losses, 'r', label='Validation Loss')
plt.title('Training and Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()

# Plot Accuracy
plt.subplot(1, 2, 2)
plt.plot(epochs, train_accuracies, 'b', label='Training Accuracy')
plt.plot(epochs, val_accuracies, 'r', label='Validation Accuracy')
plt.title('Training and Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()

plt.tight_layout()
plt.show()
```

```
# Call the function to plot metrics
plot_metrics(train_losses, val_losses, train_accuracies, val_accuracies)
```



Conclusion

In this project, we successfully developed a Convolutional Neural Network (CNN) to classify histopathologic images of lymph node sections as cancerous or non-cancerous. The model achieved a validation accuracy of 92.51% and an AUC-ROC score of 91.86%, demonstrating the ability to distinguish between the two classes. The CNN architecture utilized spatial hierarchies in the image data, contributing to its performance. Our approach included data augmentation techniques such as horizontal flipping and rotation, which helped improve generalization by simulating real-world variability in the images.

Throughout the training process, the loss and accuracy metrics for both training and validation sets showed consistent improvement, indicating that the model learned effectively without significant overfitting. However, there remains potential for further enhancements. Future work could explore more complex architectures, such as deeper neural networks like ResNet or VGG, which may capture even more intricate patterns in the data. Additionally, experimenting with different hyperparameters, learning rates, and regularization techniques could yield further improvements in performance.

Overall, this project highlights the potential of deep learning models in assisting with medical diagnostics, offering a powerful tool to aid pathologists in the detection of cancerous cells. As deep learning technology advances, such models will likely become an integral part of automated diagnostic systems, contributing to timely and accurate cancer diagnoses.

In []: