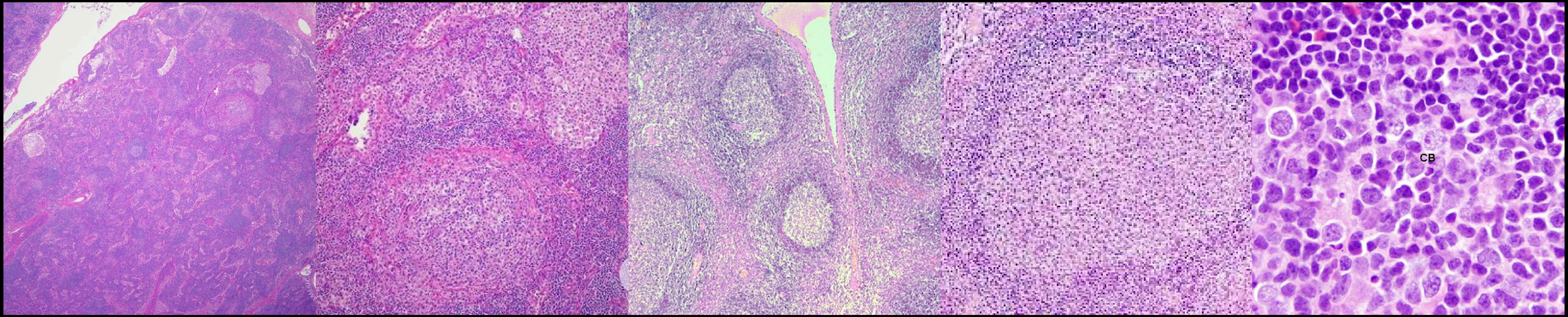
Capstone Project

Metastatic Tissue Detection (Cancer Detection)



Student: Ildar Mamin

Course: INFO-6147-(01)-25F Deep Learning with PyTorch

Date: December 09, 2025

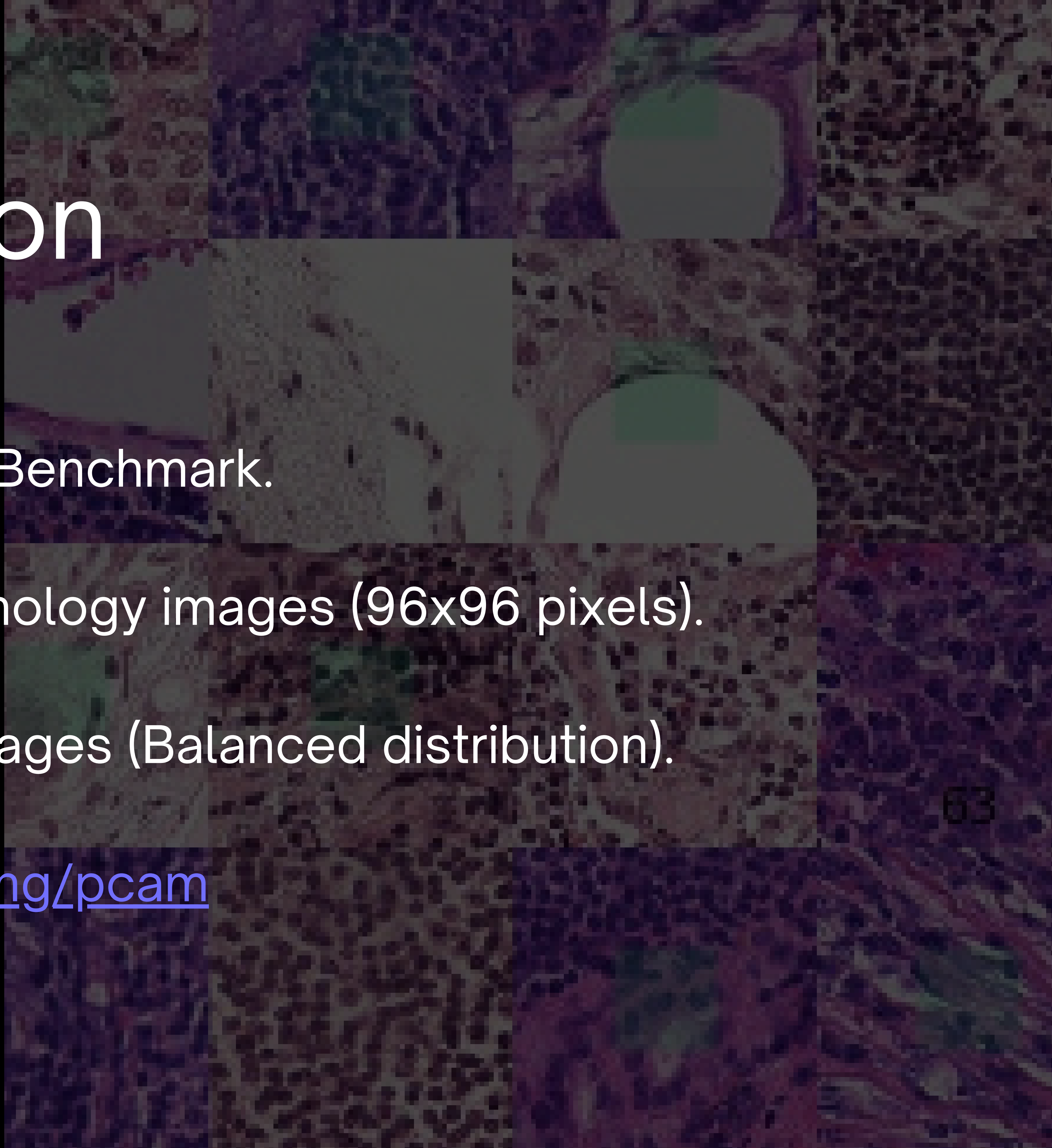
The Problem

- Current Challenge: Pathologists manually examine thousands of tissue slides.
- The Issue: Process is time-consuming, prone to fatigue, and subjective.
- The Need: Automated tools to triage and flag suspicious regions.

Project Objectives

- Primary Goal: Build a Convolutional Neural Network (CNN) to classify lymph node tissue.
- Scope: Binary Classification (Healthy vs. Tumor).
- Deliverable: A functional Web Application for real-time inference.

Dataset Description



- Source: PatchCamelyon (PCAM) Benchmark.
- Data Type: H&E stained histopathology images (96x96 pixels).
- Sample Size: Subset of 5,000 images (Balanced distribution).

Source: https://github.com/basveeling/pcam

Project structure

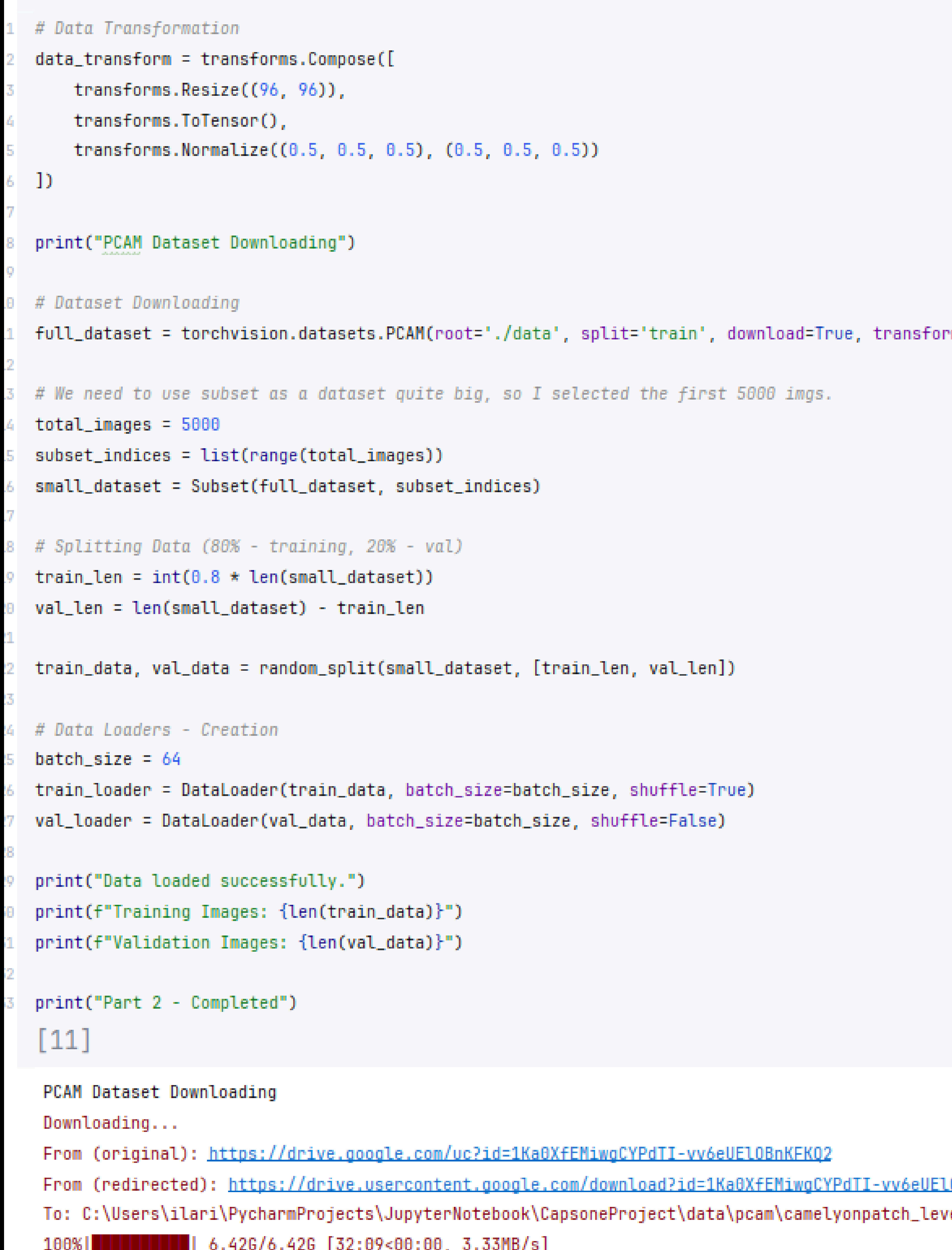
Main tools:

Brief Project Structure

- training.ipynb
- app.py
- data

Data Preprocessing

- Resizing: Standardized to 96x96.
- Normalization: Scaled pixel values (Mean=0.5, Std=0.5).
- Splitting: 80% Training / 20% Validation.
- Tensors: Converted to PyTorch format.



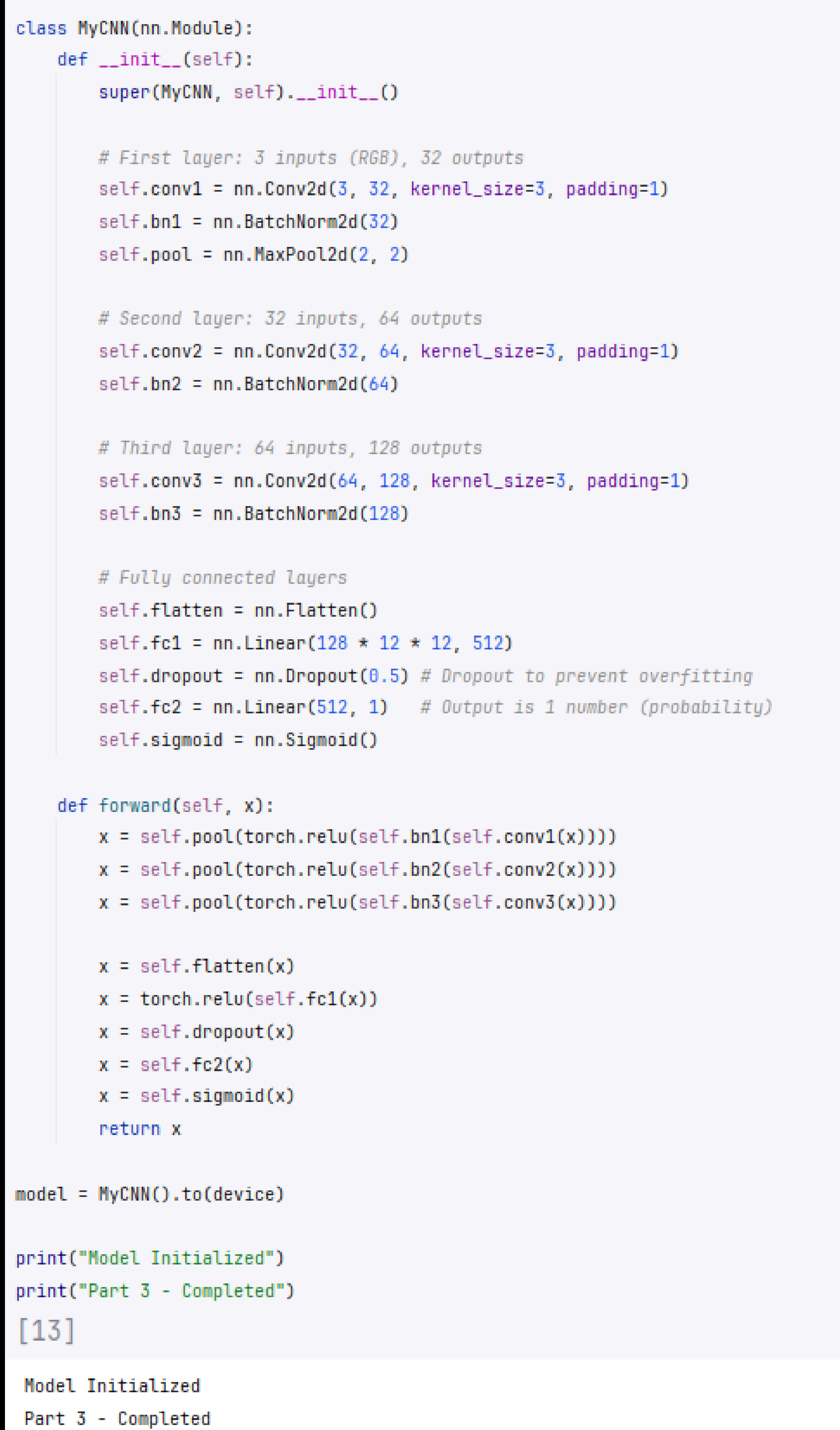
```python
# Data Transformation
data_transform = transforms.Compose([
    transforms.Resize((96, 96)),
    transforms.ToTensor(),
    transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))
])

print("PCAM Dataset Downloading")

# Dataset Downloading
full_dataset = torchvision.datasets.PCAM(root='./data', split='train', download=True, transfor

# We need to use subset as a dataset quite big, so I selected the first 5000 imgs.
total_images = 5000
subset_indices = list(range(total_images))
small_dataset = Subset(full_dataset, subset_indices)

# Splitting Data (80% - training, 20% - val)
train_len = int(0.8 * len(small_dataset))
val_len = len(small_dataset) - train_len

train_data, val_data = random_split(small_dataset, [train_len, val_len])

# Data Loaders - Creation
batch_size = 64
train_loader = DataLoader(train_data, batch_size=batch_size, shuffle=True)
val_loader = DataLoader(val_data, batch_size=batch_size, shuffle=False)

print("Data loaded successfully.")
print(f"Training Images: {len(train_data)}")
print(f"Validation Images: {len(val_data)}")

print("Part 2 - Completed")
```

[11]

```
PCAM Dataset Downloading
Downloading...
From (original): https://drive.google.com/uc?id=1Ka0XfEMiwgCYPdTI-vv6eUEl0BnKFKQ2
From (redirected): https://drive.usercontent.google.com/download?id=1Ka0XfEMiwgCYPdTI-vv6eUEl
To: C:\Users\ilari\PycharmProjects\JupyterNotebook\CapsoneProject\data\pcam\camelyonpatch_leve
100%|██████████| 6.42G/6.42G [32:09<00:00, 3.33MB/s]
```

Model Architecture

- Architecture: Custom 3-Layer CNN.
- Feature Extraction: 3 Convolutional Blocks + ReLU + MaxPool.
- Regularization: Dropout (0.5) and Batch Normalization.
- Output: Sigmoid Activation (Probability Score).



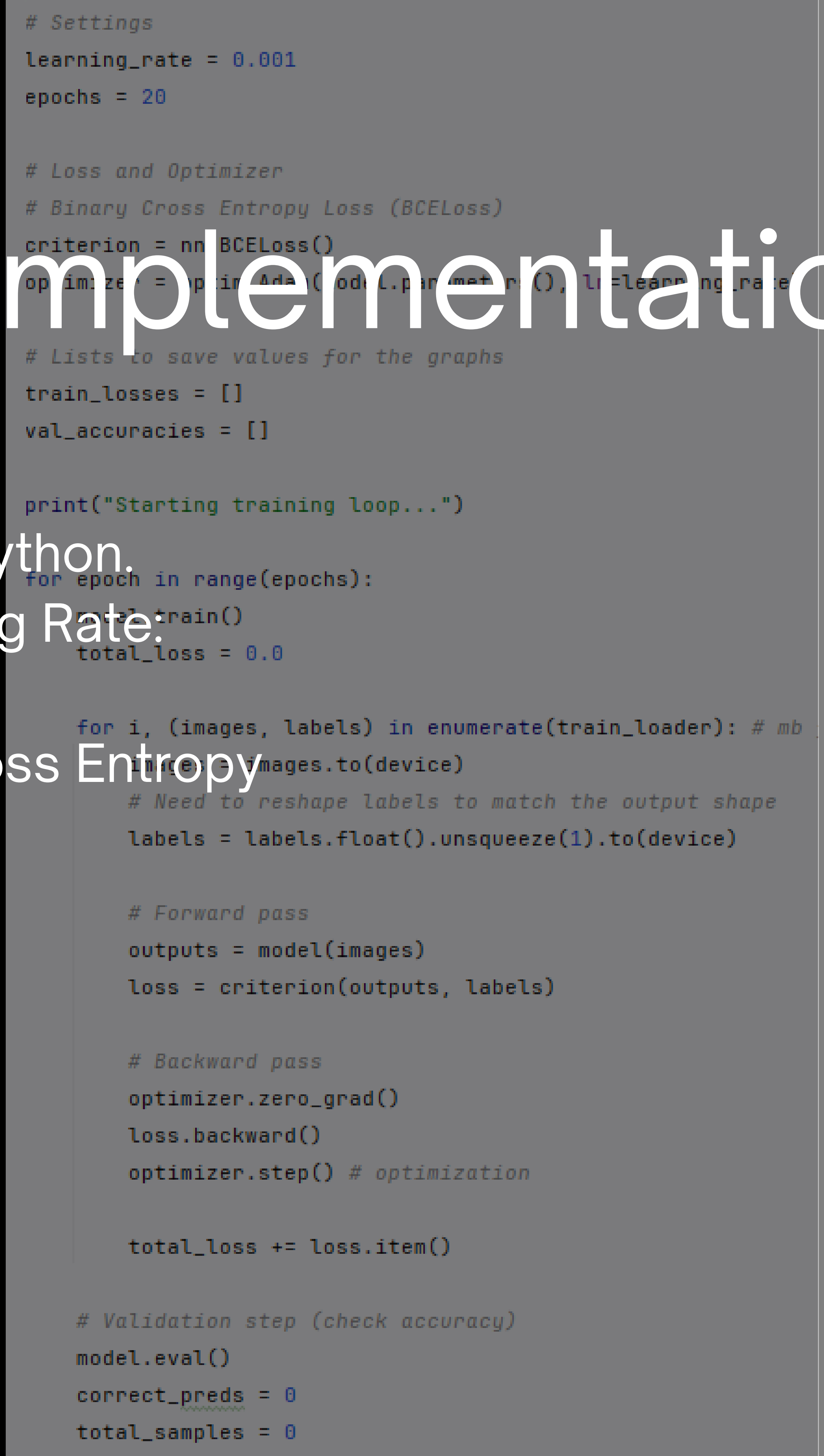
```python
class MyCNN(nn.Module):
    def __init__(self):
        super(MyCNN, self).__init__()

        # First layer: 3 inputs (RGB), 32 outputs
        self.conv1 = nn.Conv2d(3, 32, kernel_size=3, padding=1)
        self.bn1 = nn.BatchNorm2d(32)
        self.pool = nn.MaxPool2d(2, 2)

        # Second layer: 32 inputs, 64 outputs
        self.conv2 = nn.Conv2d(32, 64, kernel_size=3, padding=1)
        self.bn2 = nn.BatchNorm2d(64)

        # Third layer: 64 inputs, 128 outputs
        self.conv3 = nn.Conv2d(64, 128, kernel_size=3, padding=1)
        self.bn3 = nn.BatchNorm2d(128)

        # Fully connected layers
        self.flatten = nn.Flatten()
        self.fc1 = nn.Linear(128 * 12 * 12, 512)
        self.dropout = nn.Dropout(0.5) # Dropout to prevent overfitting
        self.fc2 = nn.Linear(512, 1)   # Output is 1 number (probability)
        self.sigmoid = nn.Sigmoid()

    def forward(self, x):
        x = self.pool(torch.relu(self.bn1(self.conv1(x))))
        x = self.pool(torch.relu(self.bn2(self.conv2(x))))
        x = self.pool(torch.relu(self.bn3(self.conv3(x))))

        x = self.flatten(x)
        x = torch.relu(self.fc1(x))
        x = self.dropout(x)
        x = self.fc2(x)
        x = self.sigmoid(x)
        return x

model = MyCNN().to(device)

print("Model Initialized")
print("Part 3 - Completed")
```

[13]

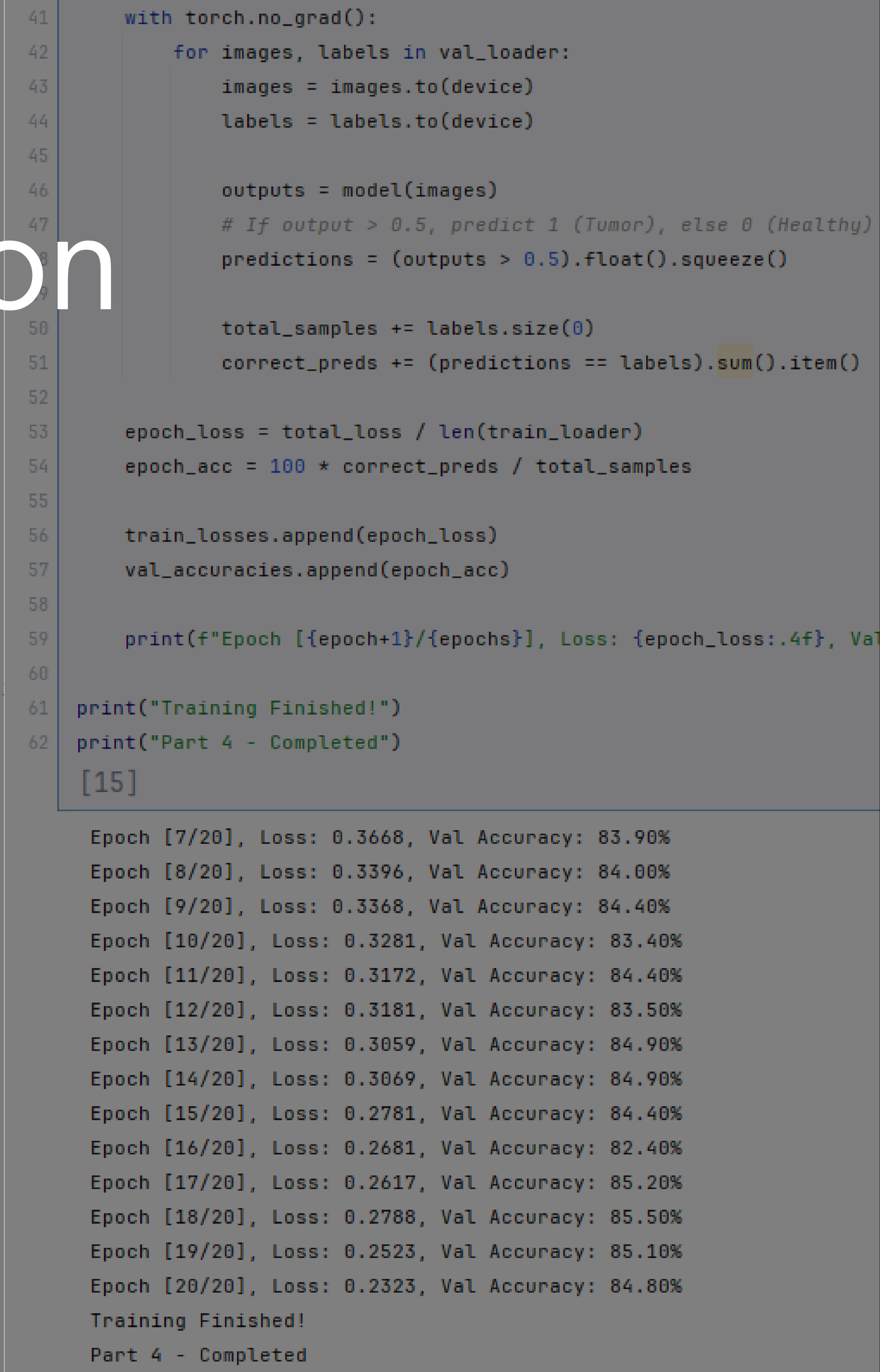
```
Model Initialized
Part 3 - Completed
```

Training & Implementation

- Framework: PyTorch & Python.
- Optimizer: Adam (Learning Rate: 0.001).
- Loss Function: Binary Cross Entropy (BCELoss).
- Epochs: 20



```python
# Settings
learning_rate = 0.001
epochs = 20

# Loss and Optimizer
# Binary Cross Entropy Loss (BCELoss)
criterion = nn.BCELoss()
optimizer = optim.Adam(model.parameters(), lr=learning_rate

# Lists to save values for the graphs
train_losses = []
val_accuracies = []

print("Starting training loop...")

for epoch in range(epochs):
    model.train()
    total_loss = 0.0

    for i, (images, labels) in enumerate(train_loader): # mb
        images = images.to(device)
        # Need to reshape labels to match the output shape
        labels = labels.float().unsqueeze(1).to(device)

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

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

        total_loss += loss.item()

    # Validation step (check accuracy)
    model.eval()
    correct_preds = 0
    total_samples = 0
```



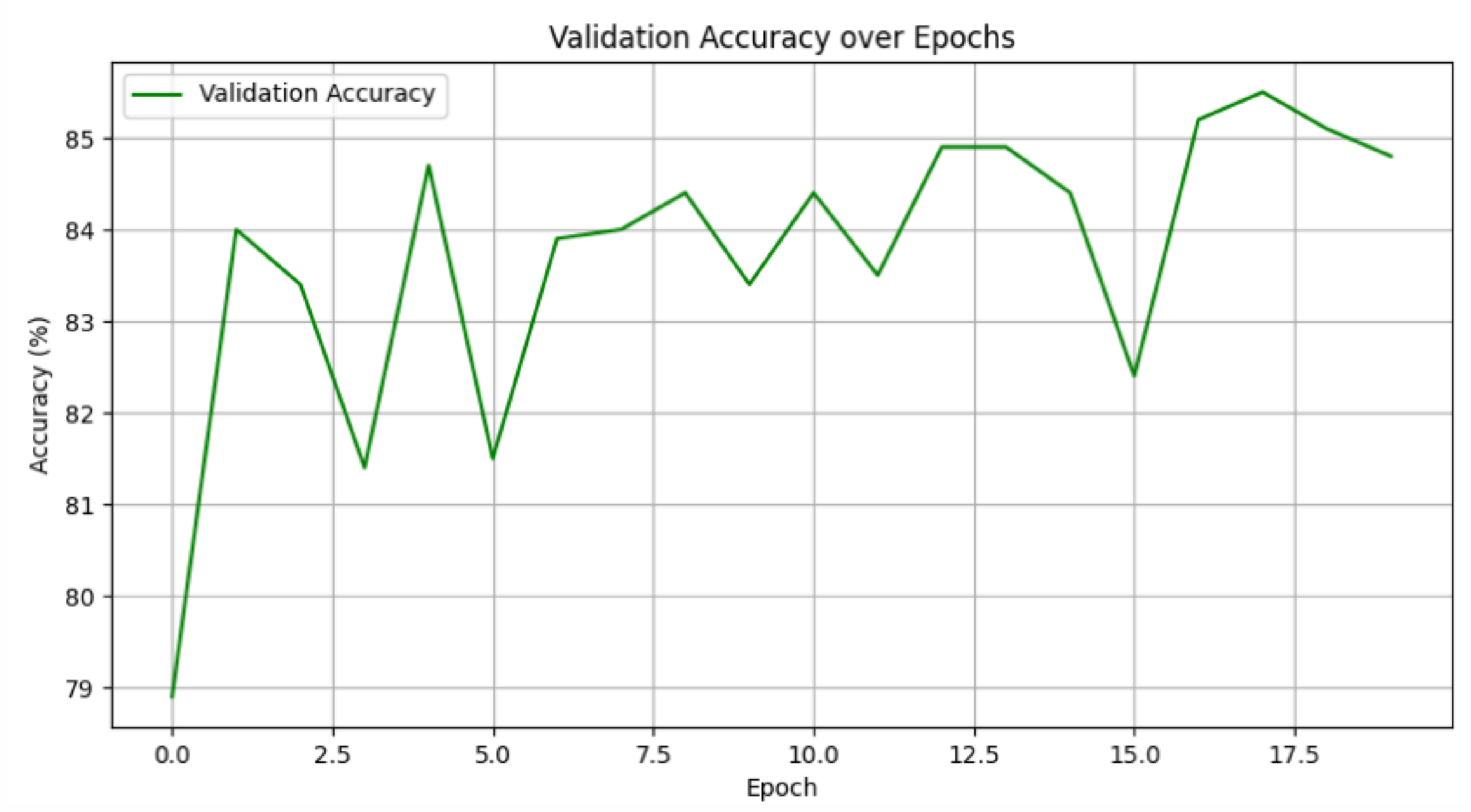
```python
    with torch.no_grad():
        for images, labels in val_loader:
            images = images.to(device)
            labels = labels.to(device)

            outputs = model(images)
            # If output > 0.5, predict 1 (Tumor), else 0 (Healthy)
            predictions = (outputs > 0.5).float().squeeze()

            total_samples += labels.size(0)
            correct_preds += (predictions == labels).sum().item()

    epoch_loss = total_loss / len(train_loader)
    epoch_acc = 100 * correct_preds / total_samples

    train_losses.append(epoch_loss)
    val_accuracies.append(epoch_acc)

    print(f"Epoch [{epoch+1}/{epochs}], Loss: {epoch_loss:.4f}, Val

print("Training Finished!")
print("Part 4 - Completed")
```

[15]

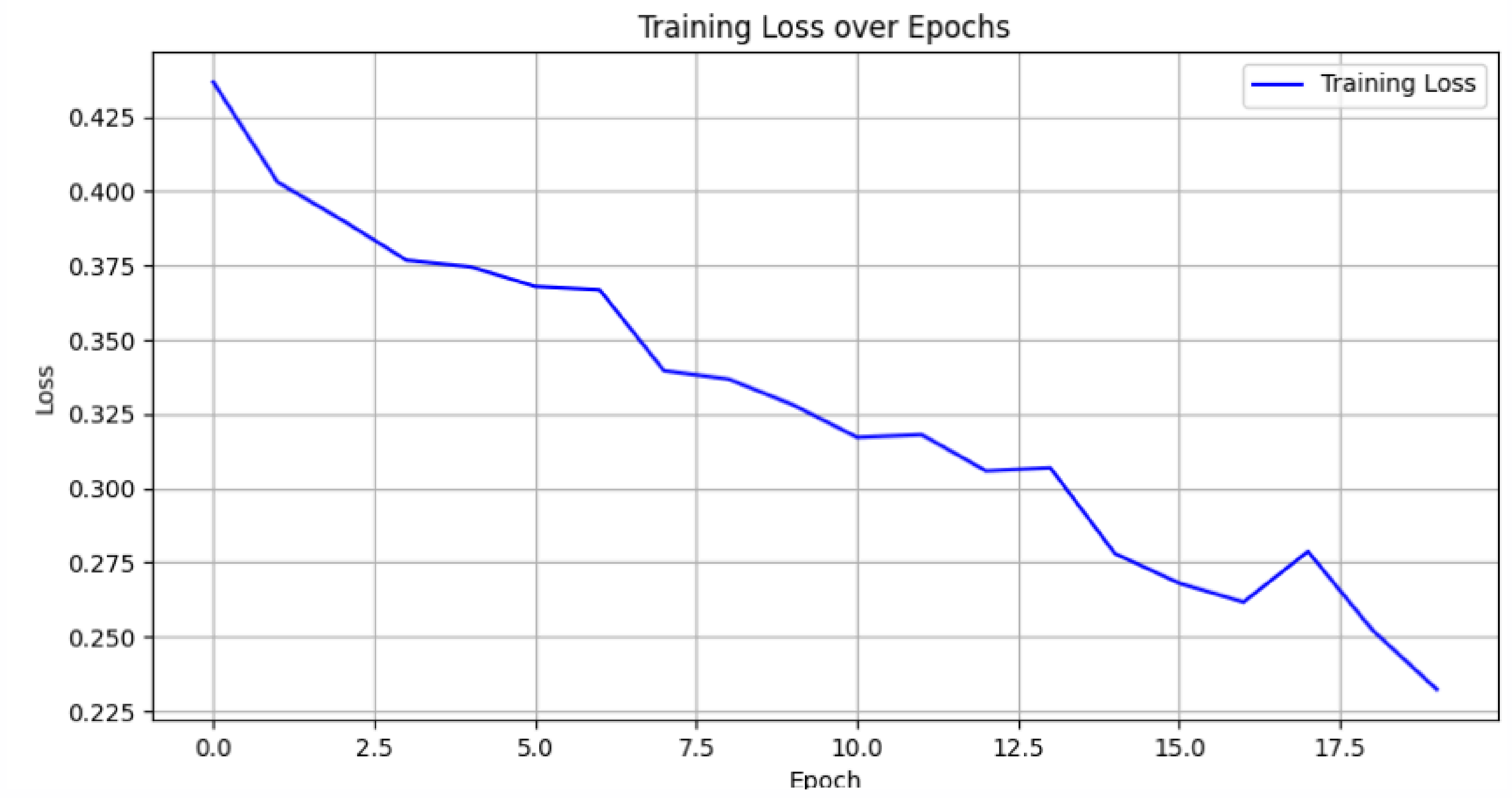
```
Epoch [7/20], Loss: 0.3668, Val Accuracy: 83.90%
Epoch [8/20], Loss: 0.3396, Val Accuracy: 84.00%
Epoch [9/20], Loss: 0.3368, Val Accuracy: 84.40%
Epoch [10/20], Loss: 0.3281, Val Accuracy: 83.40%
Epoch [11/20], Loss: 0.3172, Val Accuracy: 84.40%
Epoch [12/20], Loss: 0.3181, Val Accuracy: 83.50%
Epoch [13/20], Loss: 0.3059, Val Accuracy: 84.90%
Epoch [14/20], Loss: 0.3069, Val Accuracy: 84.90%
Epoch [15/20], Loss: 0.2781, Val Accuracy: 84.40%
Epoch [16/20], Loss: 0.2681, Val Accuracy: 82.40%
Epoch [17/20], Loss: 0.2617, Val Accuracy: 85.20%
Epoch [18/20], Loss: 0.2788, Val Accuracy: 85.50%
Epoch [19/20], Loss: 0.2523, Val Accuracy: 85.10%
Epoch [20/20], Loss: 0.2323, Val Accuracy: 84.80%
Training Finished!
Part 4 - Completed
```

Results & Evaluation

Accuracy

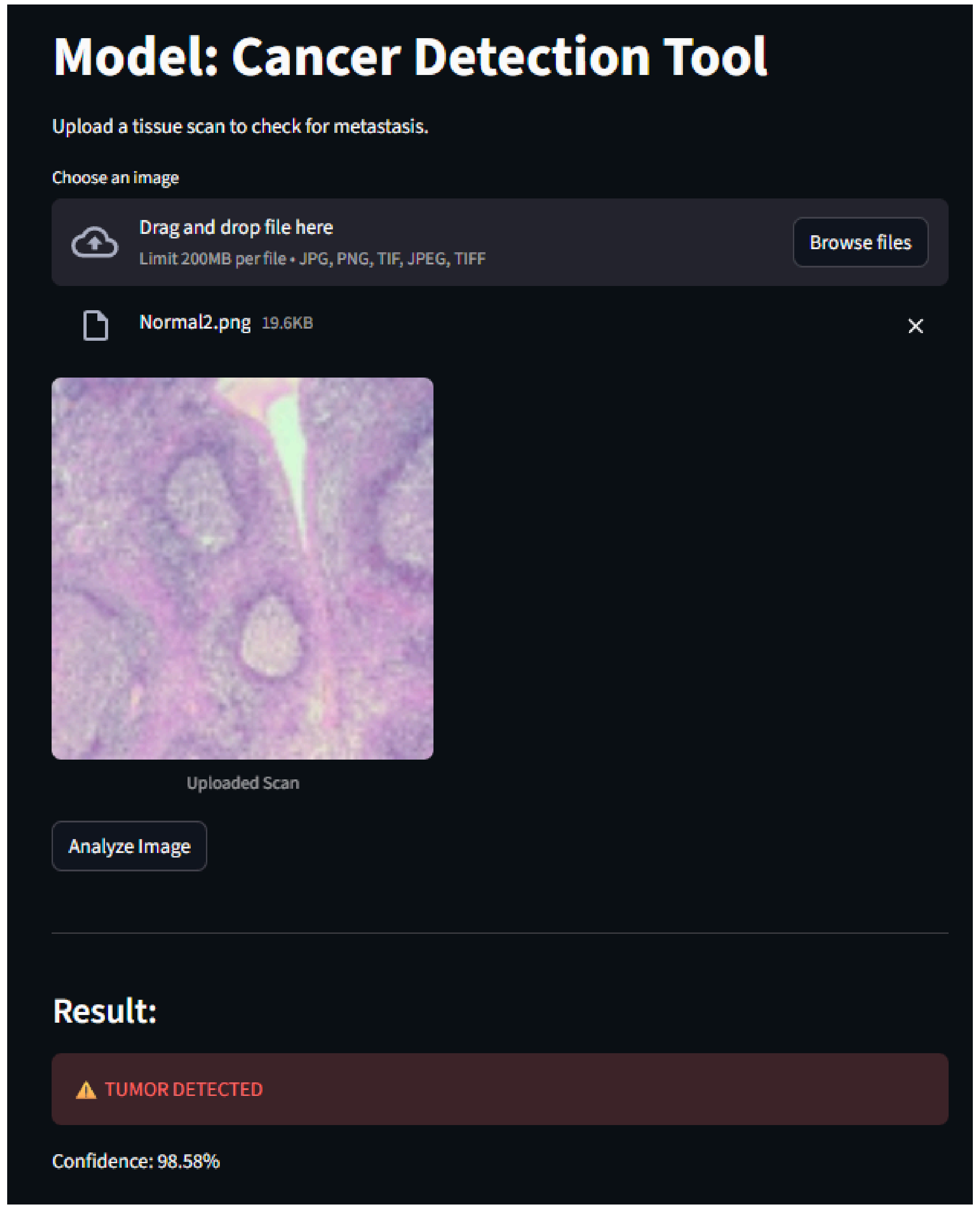


Loss



Deployment (The App)

Possitive acc 98%



Model: Cancer Detection Tool

Upload a tissue scan to check for metastasis.

Choose an image

Drag and drop file here
Limit 200MB per file • JPG, PNG, TIF, JPEG, TIFF

Browse files

Normal2.png 19.6KB

Uploaded Scan

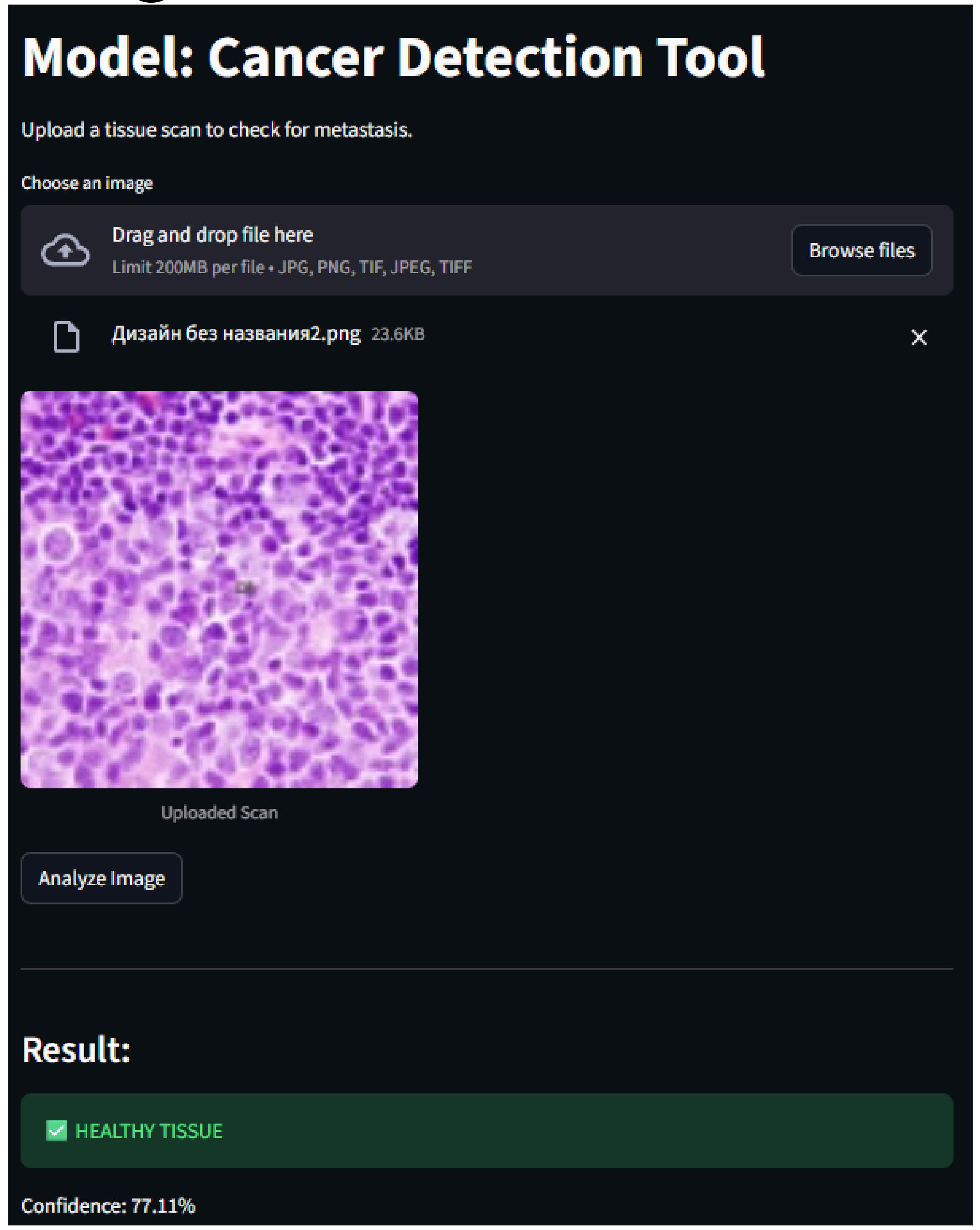
Analyze Image

Result:

⚠ TUMOR DETECTED

Confidence: 98.58%

Neg acc 77%



Model: Cancer Detection Tool

Upload a tissue scan to check for metastasis.

Choose an image

Drag and drop file here
Limit 200MB per file • JPG, PNG, TIF, JPEG, TIFF

Browse files

Дизайн без названия2.png 23.6KB

Uploaded Scan

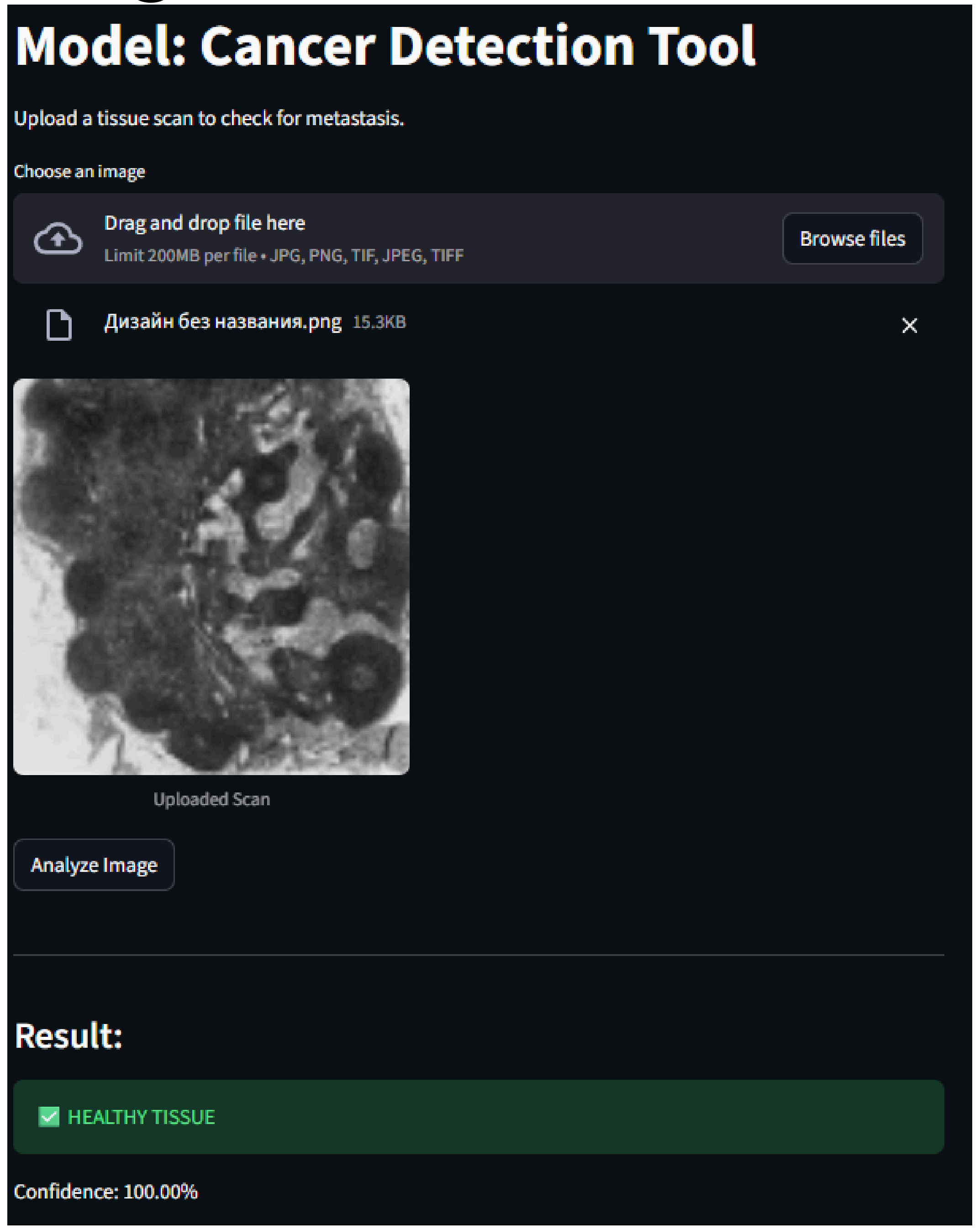
Analyze Image

Result:

✓ HEALTHY TISSUE

Confidence: 77.11%

Neg acc 99%



Model: Cancer Detection Tool

Upload a tissue scan to check for metastasis.

Choose an image

Drag and drop file here
Limit 200MB per file • JPG, PNG, TIF, JPEG, TIFF

Browse files

Дизайн без названия.png 15.3KB

Uploaded Scan

Analyze Image

Result:

✓ HEALTHY TISSUE

Confidence: 100.00%

Conclusion

- Conclusion: Successfully automated metastasis detection with high accuracy.
- Limitations: Trained on a subset; no heavy data augmentation.

Possible Improvements:

- Train on full dataset.
- Implement Transfer Learning (ResNet/VGG).
- Advanced Augmentation (Rotation/Flipping)