Report

BIAI

Multiclass classification of histopathological images of lung and colon tissue

18.06.2024

Autor:

Joanna Wiekiera

Maryna Żur

1. Introduction

Histopathological image analysis is crucial in modern medical diagnostics, providing insights into tissue characteristics that aid in disease classification and treatment decisions. This project focuses on the multiclass classification of histopathological images from lung and colon tissues, aiming to distinguish between five distinct classes:

- lung benign tissue
- lung adenocarcinoma
- lung squamous cell carcinoma
- colon adenocarcinoma
- colon benign tissue.

2. Analysis of the task

2.1. Solve the problem

The goal of our project was to accurately differentiate between five distinct classes: lung benign tissue, lung adenocarcinoma, lung squamous cell carcinoma, colon adenocarcinoma, and colon benign tissue. By leveraging convolutional neural networks, which excel at learning hierarchical features from image data, we aim to develop a robust model capable of accurately categorizing these histopathological images. This classification task is essential diagnostic accuracy and facilitating more informed medical decisions based on detailed tissue analysis.

2.2. Dataset

The dataset employed comprises 25,000 high-resolution images, each measuring 768 x 768 pixels and stored in JPEG format. These images capture the intricate details of tissue samples, presenting both normal and cancerous conditions across lung and colon tissues. Class representatives are shown in Figure 1.

The original sample consisted of:

- 750 total images of lung tissue:
  - 250 benign lung tissue
  - 250 lung adenocarcinomas
  - 250 lung squamous cell carcinomas
- 500 total images of colon tissue:
  - 250 benign colon tissue
  - 250 colon adenocarcinomas

This original dataset was augmented to 25,000 using the Augmentor package by the dataset author. Equal class sizes resulted in better performance.

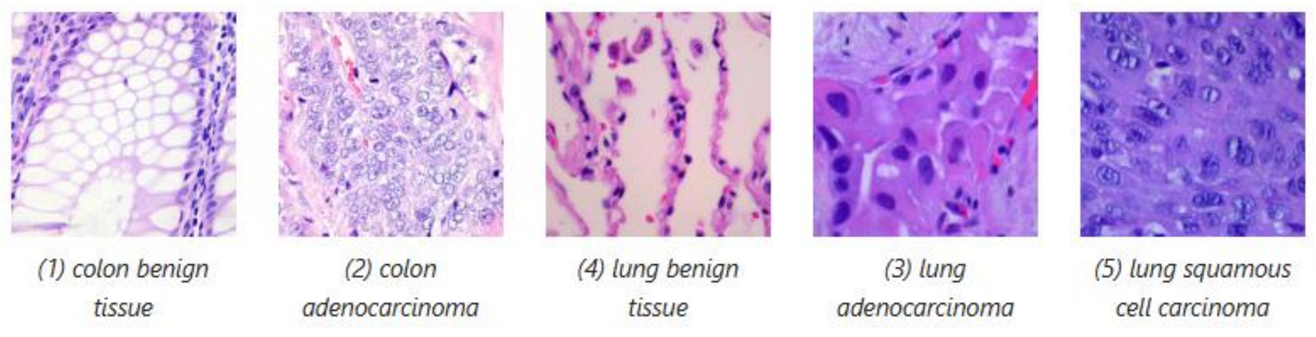


Figure 1: The types of classes in the dataset

2.3. Using tools

Our project uses PyTorch to train our convolutional neural network (CNN) model due to its robust capabilities in deep learning research and development. PyTorch provides a flexible and intuitive platform for building and optimizing neural networks, making it ideal for our task of histopathological image classification.

To document and describe the results of our training process, we employ Jupyter Notebook. Jupyter Notebook is a web-based interactive environment that allows us to create and share documents containing live code, equations, visualizations, and narrative text. It enables us to present our experimental setup, code implementation, training progress, and evaluation metrics clearly and organized.

3. Internal and external specification of software solution

In this part of the report, we describe the various components and features of our software solution.

3.1. Internal specification

In this part of specification we describe the models, functions and methods used in our project.

a) Data preprocessing

The data preprocessing pipeline for our dataset involves several steps.

Firstly, as shown in Figure 2, we defined a series of transformations to be applied to each image in the dataset:

i) Resize – each image was resized to the dimension of 64 x 64 pixels using transforms.Resize((64, 64)) function

ii) Random Horizontal Flip – To augment the dataset and introduce variability, each image has a 50% chance of being horizontally flipped using the transforms.RandomHorizontalFlip() function

iii) Tensor – The images are converted to PyTorch tensors using the transforms.ToTensor() function. This conversion is necessary for the images to be compatible with PyTorch's neural network modules.

```
# Define transforms
transforms = transforms.Compose([
    transforms.Resize((64, 64)),
    transforms.RandomHorizontalFlip(),
    transforms.ToTensor()
])
```

Listing 1: Transformation of images in dataset

Secondly, we load and split the dataset into three subsets: training (70%), validation (10%), and test set (20%). This step is shown in Figure 3.

```
# Load the dataset
dataset = datasets.ImageFolder("../DATA/lung_colon_image_set",
transform=transforms)
# Split the dataset into training and testing sets
train_size = int(0.7 * len(dataset))  # 70% for training
val_size = int(0.1 * len(dataset))  # 10% for validation
test_size = len(dataset) - train_size - val_size  # 20% for
testing
train_dataset,        val_dataset,        test_dataset        =
random_split(dataset, [train_size, val_size, test_size])
```

Listing 2: Splitting the data into training, validation, and test sets

Data loaders are created for each subset to facilitate efficient batch processing during training and evaluation. The data loaders handle the batching of images and shuffling of the training data to improve the model's learning.

```
# Create DataLoaders
train_loader   =   DataLoader(train_dataset,   batch_size=64,
shuffle=True)
val_loader    =    DataLoader(val_dataset,    batch_size=64,
shuffle=False)
test_loader    =    DataLoader(test_dataset,    batch_size=64,
shuffle=False)
```

Listing 3: Creating DataLoaders for training, validation, and testing sets

b) Class Net

Defines the architecture of the convolutional neural network (CNN) employed for image classification. It includes a feature extractor composed of convolutional and pooling layers designed to capture hierarchical features from input images. Subsequently, a classifier module consisting of fully connected layers with activation functions ReLU and dropout layers for regularization purposes follows. The network was trained and optimized using these components, achieving superior performance compared to previously trained models.

```
# Define the neural network
class Net(nn.Module):
    def __init__(self, num_classes):
        super(Net, self).__init__()
        self.feature_extractor = nn.Sequential(
            nn.Conv2d(3, 32, kernel_size=3, padding=1),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2),  # Output: 32x32x32

            nn.Conv2d(32, 64, kernel_size=3, padding=1),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2),  # Output: 64x16x16

            nn.Conv2d(64, 128, kernel_size=3, padding=1),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2),  # Output: 128x8x8

            nn.Flatten()
        )
        self.classifier = nn.Sequential(
            nn.Linear(128*8*8, 256),
            nn.ReLU(),
            nn.Dropout(0.2),
            nn.Linear(256, num_classes)
            )
```

Listing 4: The structure of our model

c) Function and validation loop

We established a training and validation loop that implemented the CNN training process using stochastic gradient descent (SGD) or the Adam optimizer. This loop computed the loss, executed backpropagation, updated model parameters, and tracked training metrics such as loss and accuracy. In the validation loop, we assessed the model's performance on a validation set after each epoch to monitor potential overfitting and selected the optimal model based on validation loss.

d) Function Evaluation

The evaluation function on the test set involves calculating the accuracy metric, which assesses the percentage of correctly predicted labels compared to the total number of samples in the test set. This metric serves as a crucial indicator of the model's overall performance on unseen data.

In our study, the accuracy on the test set reached 97.46%. This high accuracy indicates that the model successfully classified the majority of test samples correctly, demonstrating robust performance in distinguishing between different classes of histopathological images of lung and colon tissues. Such a high accuracy suggests that the model effectively learned relevant features and patterns from the training data, translating into reliable predictions on unseen data.

e) Training result on plot

We provided the result of training and validation losses and accuracies using a plot. The plotted figures illustrate the progression of training and validation metrics across epochs during the model training process. The Figure 6 represents:

- The left subplot shows the change in training loss and validation loss over each epoch.
- The right subplot displays the evolution of training accuracy and validation accuracy across epochs.

The results of our training are highly satisfactory. The analysis of training results indicates that the model effectively learned, as evidenced by decreasing loss values for both the training and validation datasets. The model’s accuracy consistently improved with each epoch, reaching high levels of 97.97% for the training set and impressive 98.28% for the validation set. While initial fluctuations in training accuracy were observed, these values stabilized as the model training progressed. This demonstrates the absence of overfitting and the effectiveness of the model learning process. The results are shown in Figure 2.

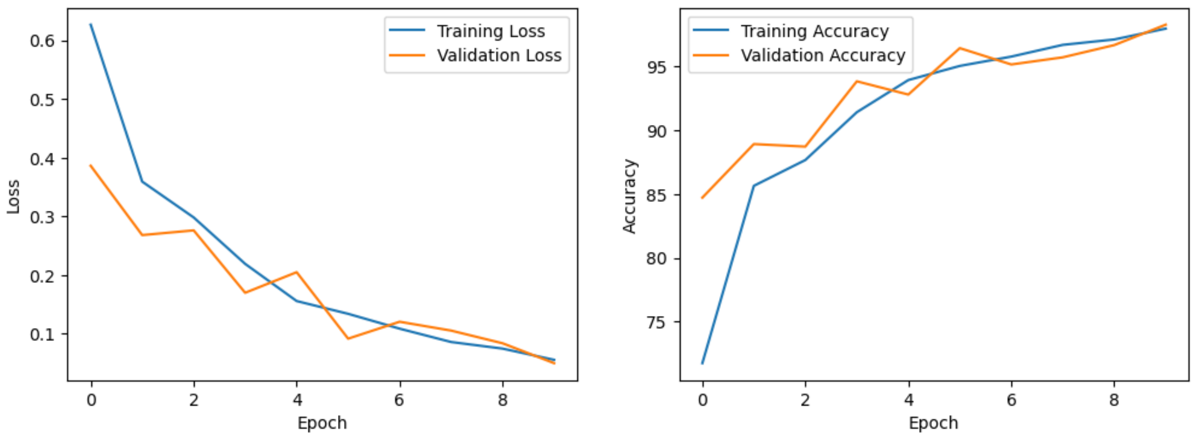


Figure 2: Training results plots of our model

f) Confusion Matrix

In Figure 3, the confusion matrix generated from the classification of the test data is shown. It is visible that images illustrating colon cancer are most often confused with healthy colon tissue and vice versa, but very rarely with lung tissue. At the same time, diseased lung tissues are confused with each other but are rarely mistaken for healthy lung tissue, which, in the context of medical diagnostics, is the safest outcome. It is crucial to avoid situations where diseased patients are diagnosed as healthy.

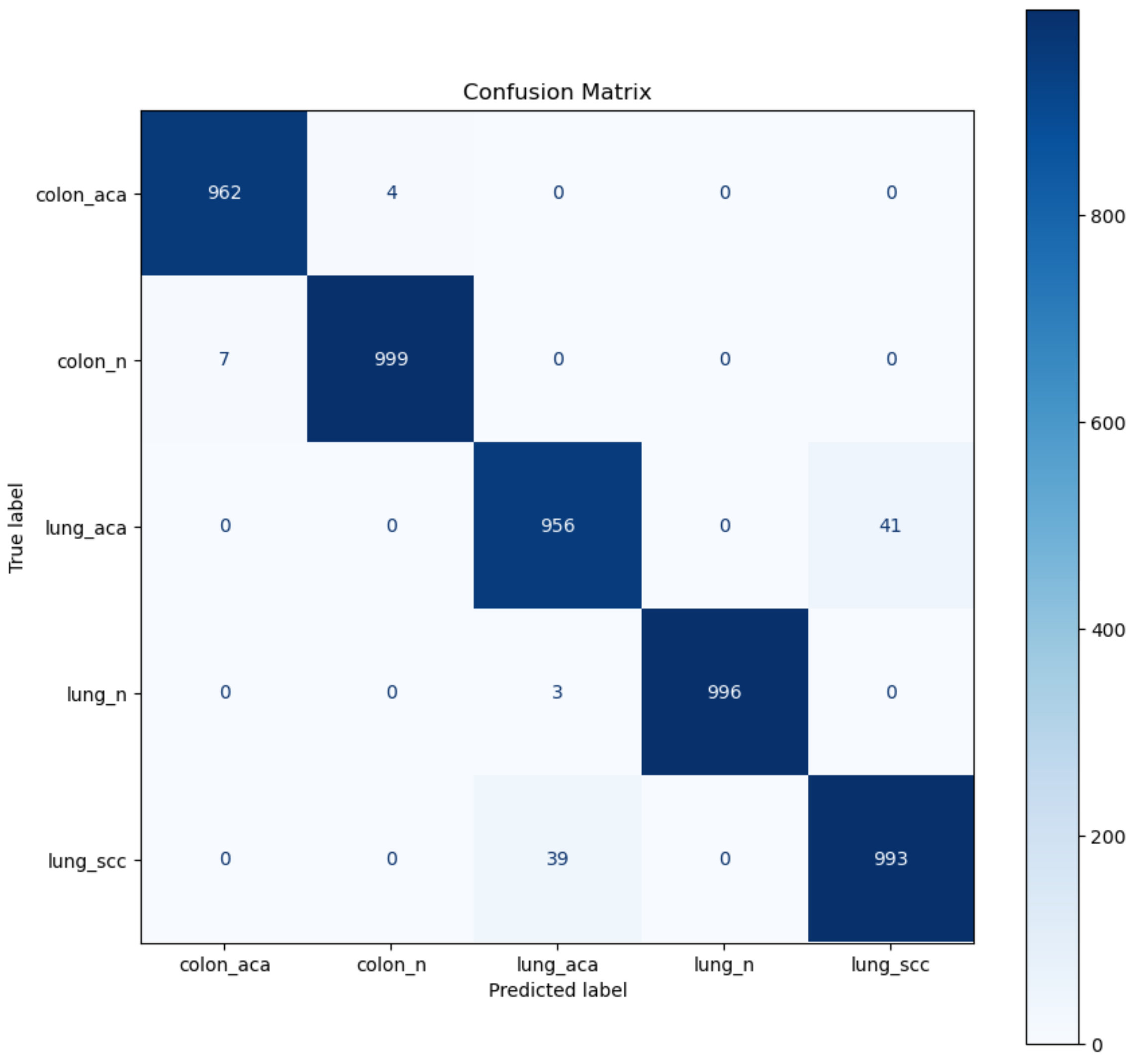


Figure 3: The types of classes in the dataset

3.2. External specification

We used Jupyter Notebook, which serves as the primary interface for documenting the project, executing code cells, and visualizing results such as training/validation losses and accuracies using Matplotlib. We used tqdm library to show the progress of the training and validation loop.

4. Experiments

Initially, our convolutional neural network consisted of two convolutional layers and a single linear classification layer. However, the learning results from this model were unsatisfactory. Therefore, we decided to enhance the model by adding convolutional layers and restructuring the entire classification function. This involved incorporating extra layers and a dropout function.

Our first model is shown in Listing 5. It has two convolutional layers with ELU activation functions and one linear layer in the classification part of the model.

```
# Define the neural network
class Net(nn.Module):
    def __init__(self, num_classes):
        super(Net, self).__init__()
        self.feature_extractor = nn.Sequential(
            nn.Conv2d(3, 32, kernel_size=3, padding=1),
            nn.ELU(),
            nn.MaxPool2d(kernel_size=2),  # Output: 32x64x64

            nn.Conv2d(32, 64, kernel_size=3, padding=1),
            nn.ELU(),
            nn.MaxPool2d(kernel_size=2),  # Output: 64x32x32

            nn.Flatten()
        )
        self.classifier = nn.Linear(64*16*16, num_classes)
```

Listing 5: The structure of our model

We analyzed the plot of loss and accuracy, and the results are presented in Figure 4. During the training, our accuracy and loss function were similar and close to 100%, leading us to question the validity of our model.

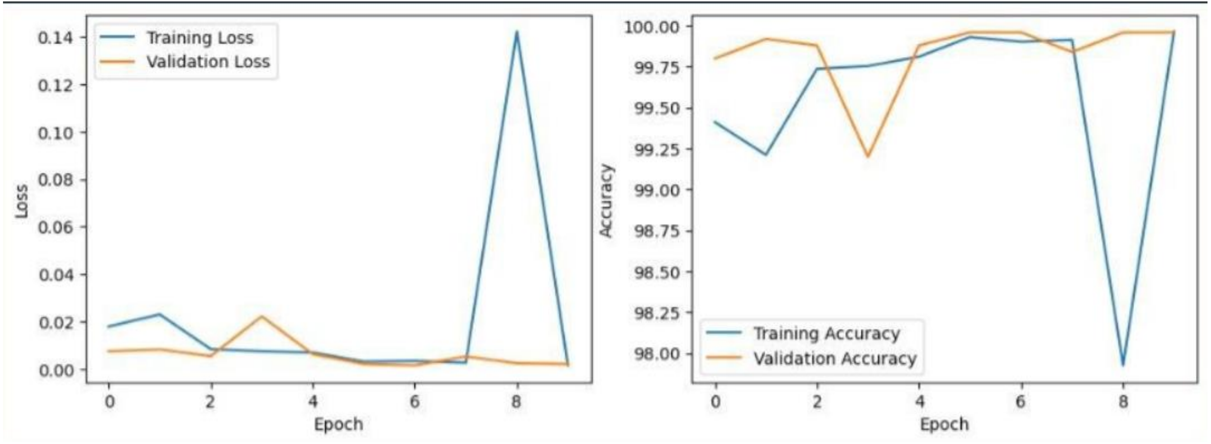


Figure 4: Training results plots of our model

The accuracy of our model was very high, leading us to consider the possibility of overfitting in our CNN. We created an SVM model to compare its accuracy with our CNN, and the SVM model also performed well. This suggests that our data might be too easy. The SVM model is presented in Listing 6.

```
# Definition and training of the SVM model
svm_model = SVC(kernel='linear', C=1)
svm_model.fit(X_train, y_train)
# Prediction on the test set
y_pred = svm_model.predict(X_test)
# Model evaluation
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy on the test set: {accuracy * 100:.2f}%')
```

Listing 6: The structure of SVM model

Hoveover, at the beginning of creating our model, we noticed alarmingly high results in the early epochs of network training. This could indicate that the task complexity was too simple for the method we were using. After a thorough review of the dataset, we found that its folders were incorrectly organized, resulting in binary classification instead of the intended five-class classification. After correcting the data loading, the model showed the expected outcomes. The result after changing looks like on Figure 5. The accuracy of our model was 92.88%.

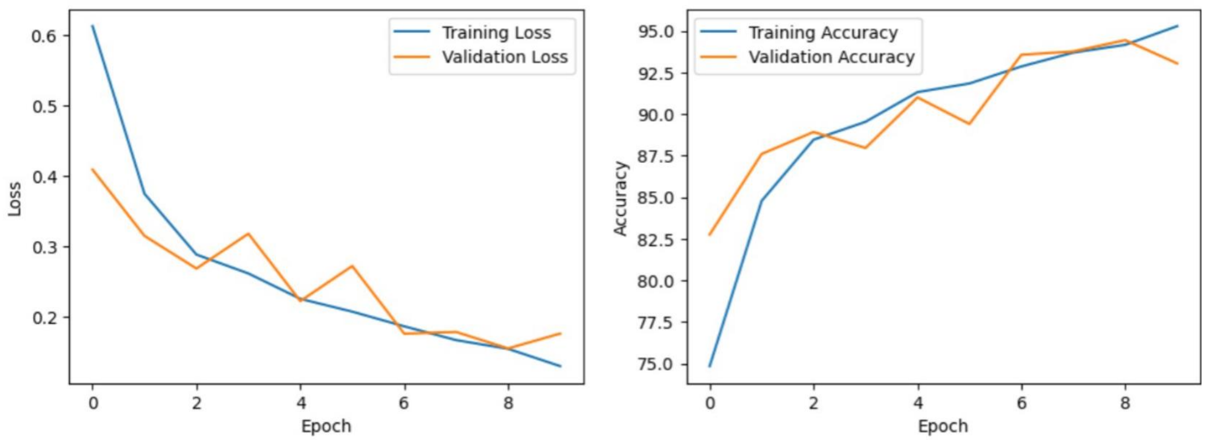


Figure 5: Training results plots of our model

The result of our model wasn't good enough for us so we started to change some components in our CNN structure. The models look like on the Listing 7 and they had accuracy 95.56%.

```
# Define the neural network
class Net(nn.Module):
    def __init__(self, num_classes):
        super(Net, self).__init__()
        self.feature_extractor = nn.Sequential(
            nn.Conv2d(3, 32, kernel_size=3, padding=1),
            nn.ELU(),
            nn.MaxPool2d(kernel_size=2),  # Output: 32x32x32
            nn.Conv2d(32, 64, kernel_size=3, padding=1),
            nn.ELU(),
            nn.MaxPool2d(kernel_size=2),  # Output: 64x16x16

            nn.Conv2d(64, 128, kernel_size=3, padding=1),
            nn.ELU(),
            nn.MaxPool2d(kernel_size=2),  # Output: 128x8x8

            nn.Flatten()
        )
        self.classifier = nn.Sequential(
            nn.Linear(128*8*8, 256),
            nn.ELU(),
            nn.Dropout(0.5),
            nn.Linear(256, num_classes)
            )
```

Listing 7: The structure of our model

While initial fluctuations in training accuracy were observed, these values stabilized as the model training progressed. This demonstrates the absence of overfitting and the effectiveness of the model learning process. The results are shown in Figure 6.

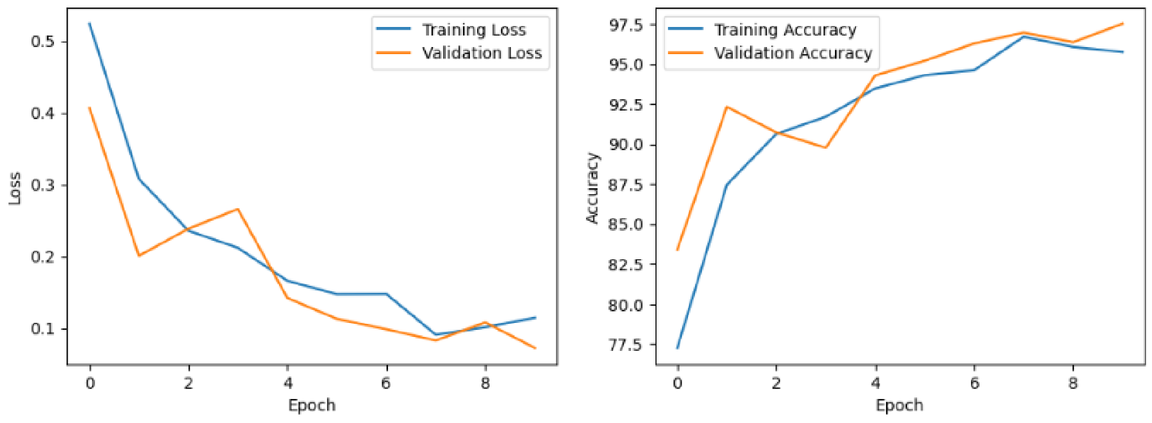


Figure 6: Training results plots of our model

Our next experiment involved changing the Dropout function's value to 0.2 in order to achieve improved results in plots and accuracy. The changes are reflected in Listing 8.

```
self.classifier = nn.Sequential(
            nn.Linear(128*8*8, 256),
            nn.ELU(),
            nn.Dropout(0.5),
            nn.Linear(256, num_classes)
            )
```

Listing 8: Implementation of Dropout with 0.2

The accuracy of our model was 96.06%, and the results are shown in the plot in Figure 7. The plots generated during training differed from earlier ones, indicating noticeable jumps in both loss and accuracy over the epochs.

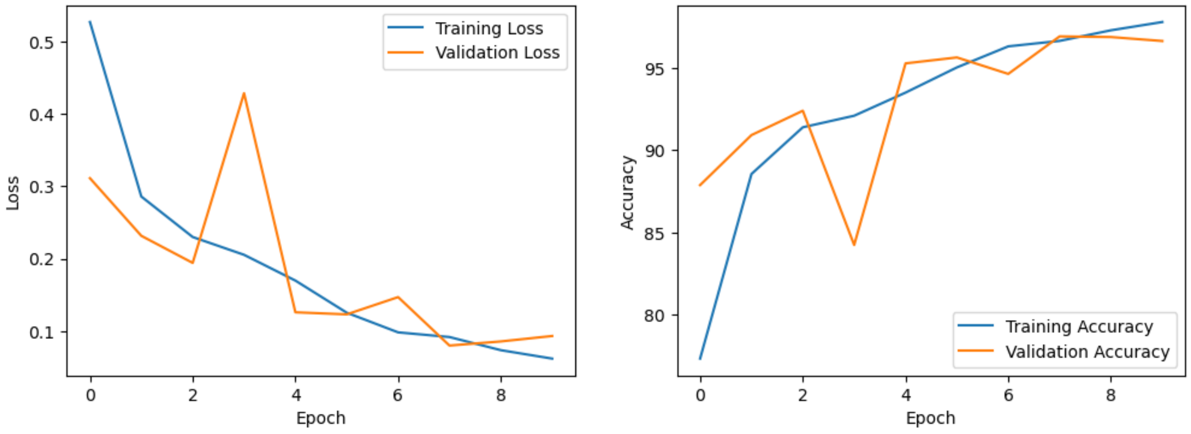
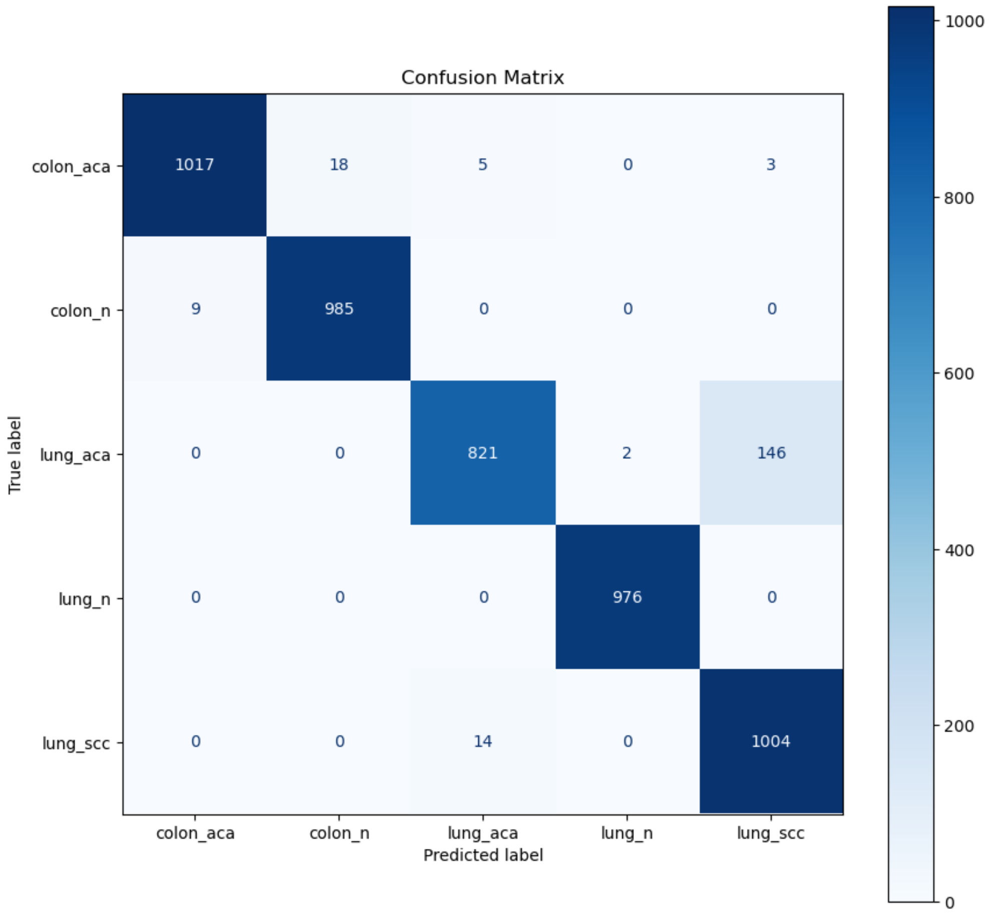


Figure 7: Training results plots after changing Dropout

Moreover, we generated the Confusion Matrix to compare the results. In Figure 8, we observe that the results are similar and the confusion matrix is the same as in the earlier version. However, the plots indicate worse performance, prompting us to consider changes to other aspects of our model structure.



True label \ Predicted label	colon_aca	colon_n	lung_aca	lung_n	lung_scc
colon_aca	1017	18	5	0	3
colon_n	9	985	0	0	0
lung_aca	0	0	821	2	146
lung_n	0	0	0	976	0
lung_scc	0	0	14	0	1004

Figure 8: Training results plots after changing Dropout

Our next step was to change the activation functions. In CNNs, it is often better to use ReLU activation functions, so we implemented this change in our project. This model performed the best, achieving an accuracy of 98.12%. The result of our training is shown in Figure 2, and the Confusion Matrix is shown in Figure 3. This model was saved in our project folder and is used in the prediction function.

5. Summary

We presented our project on the multiclass classification of histopathological images of lung and colon tissues using convolutional neural networks. We accurately differentiated between five classes and employed a dataset of 25,000. Utilizing PyTorch, we implemented a robust preprocessing pipeline and model architecture, achieving a test accuracy of 97.46%. Detailed analysis, including a confusion matrix, demonstrated effective learning and reliable performance. Our findings underscore the potential of deep learning in medical image analysis, aiding accurate diagnostics.

6. References

a) PyTorch documentation:

PyTorch documentation — PyTorch 2.3 documentation

b) Articles:

Convolutional neural networks in medical image understanding: a survey - PMC (nih.gov)

Convolutional Neural Networks, Explained | by Mayank Mishra | Towards Data Science

c) YouTube:

Neural Networks Part 8: Image Classification with Convolutional Neural Networks (CNNs)

d) DataCamp:

https://app.datacamp.com/learn/courses/intermediate-deep-learning-with-pytorch

7. GitHub

Link to GitHub repository with all files of our project:

https://github.com/jwiekiera/histopathological-classification/tree/main