Lung Cancer Diagnosis with Medical Imaging

Team: Ronald Nap

December 10, 2023

Contents

1 Introduction

1.1 Problem Statement and Dataset

The primary objective of this study is to enhance the accuracy and speed of lung cancer diagnosis using medical imaging techniques. We utilized the Lung and Colon Cancer Histopathological Image Dataset (LC25000), a comprehensive dataset containing 25,000 color images associated with five histologic entities. [1].

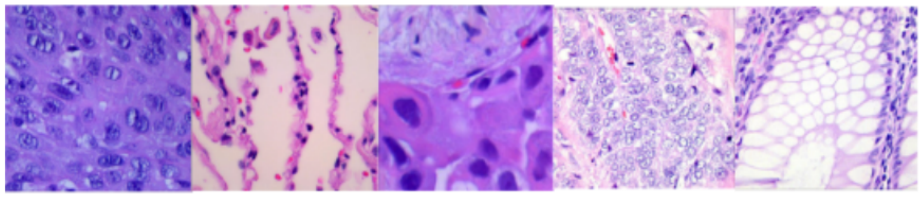


Figure 1: The first three images from the left correspond to the lung and the two images on the right correspond to the colon

1.2 Research Question and Motivation

The central research question guiding our investigation is whether machine learning and computer vision can improve the accuracy and efficiency of lung cancer diagnosis. The motivation stems from the potential to revolutionize cancer diagnosis through the application of cutting-edge technologies.

2 Dataset

2.1 Dataset Description

The dataset consists of 25,000 images categorized into five classes. Three classes are related to lung cancer: Lung Benign Tissue, Lung Adenocarcinoma, and Lung Squamous Cell Carcinoma and the other two classes are related to colon cancer: Colon Benign Tissue and Colon Adenocarcinoma. Notably, we focused only on the lung cancer tissues due to computational and time constraints. Overall, 15,000 images, with each class containing 5,000 images were used to conduct our experiments.

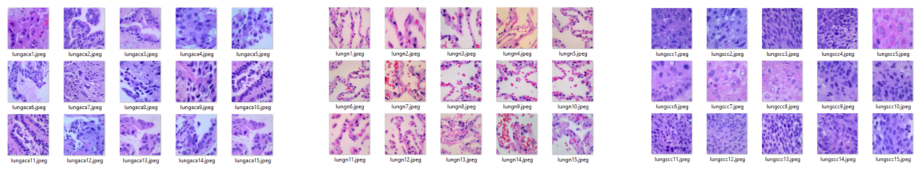
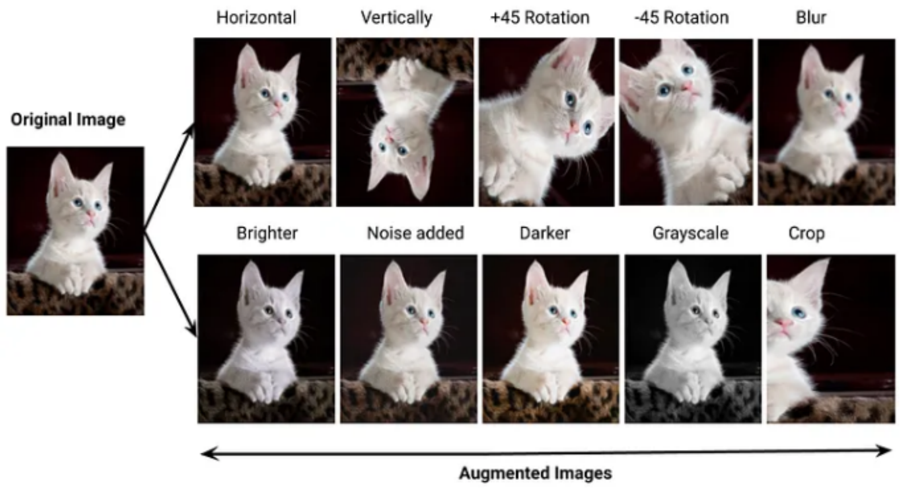


Figure 2: Here is a comprehensive view of the each class in the dataset.

2.2 Pre-processing Details

Most pre-processing steps were done by the authors of the dataset [1]. The authors originally had 250 tissues images of each class, data augmentation was performed resulting in 5,000 images per class after augmentation. The augmentations performed were by applying left and right rotations, horizontal and vertical flips with random probabilities. The authors additionally scaled the original size of the images 1024 x 768 pixels down to 768 x 768 pixels resulting in a square image. I additionally scaled down the images to 224 x 224 pixels to conserve computational resources and allow for model compatibility.



2.3 Data Exploration

Prior to diving into fitting models, we explore the dataset to gain insights into its characteristics. One aspect of this exploration involved the creation of a color histogram to assess whether the model could perform detection based on distinct color distributions.

2.3.1 Color Histogram Analysis

A color histogram is a representation of the distribution of colors in an image. By visualizing the frequency of different colors, it provides an understanding of the image's overall color composition. This exploration aimed to uncover potential patterns or distinctive color features within the histologic entity images. The color histogram was generated from the entirety of the lung cancer dataset, showcasing the distribution of pixel intensities. The average of the RGB channels was averaged across each color channel inorder to create the figure. This analysis serves as a preliminary investigation into the feasibility of the model detecting histologic entities based on unique color signatures.

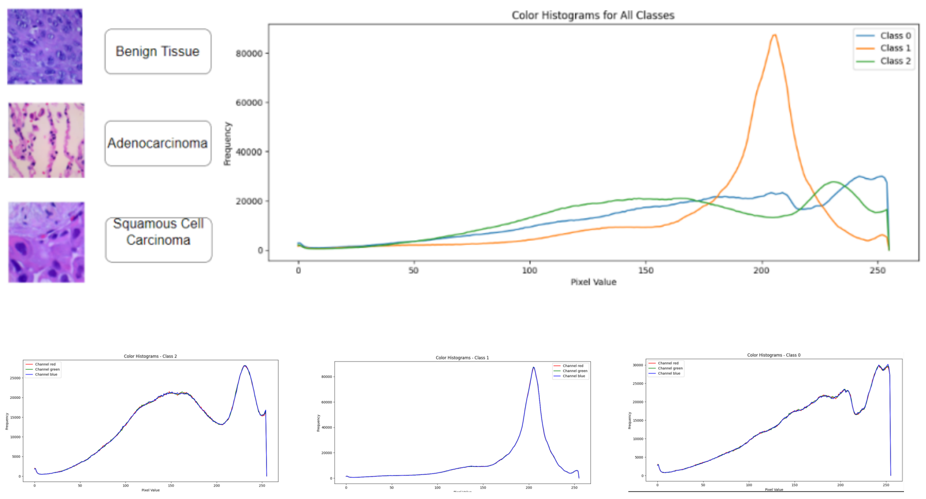
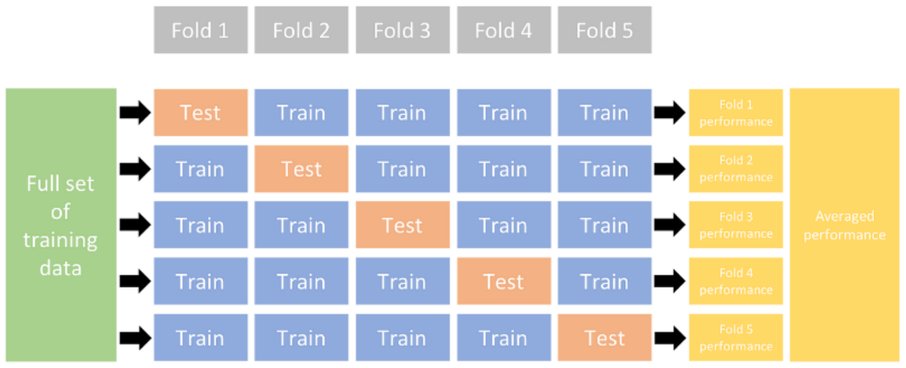


Figure 3: The color distribution of each RGB channel plotted individually was nearly identical with very minor differences hence it was decided that calculating the average RGB across each class would be most applicable.

2.4 Data Preparation

A crucial aspect of our methodology involves robust data preparation to ensure model generalization and prevent overfitting. To achieve this, we implemented 5-Fold Cross-Validation, a widely recognized technique in machine learning for model validation. The dataset was stratified and split into 5 folds, with 12,000 images allocated for training and 3,000 images for testing in each fold. The class distribution was balanced within each fold, with 4,000 images for training and 1,000 images for testing in each class. Accuracy was chosen as the primary evaluation metric due to the balanced distribution among classes.



3 Methods

3.1 Machine Learning vs Deep Learning

The choice between traditional machine learning (ML) and deep learning (DL) models involves trade-offs. Traditional ML models are valued for their interpretability and simplicity, performing well with tabular data. However, their effectiveness diminishes when handling complex, high-dimensional medical images. In contrast, deep learning models are especially effective with images, automatically learn intricate patterns and spatial hierarchies through convolutional neural networks. In the later stages of our analysis, we will fit both ML and DL models. We expect that DL models will outperform traditional ML models, showcasing superior accuracy in classifying histologic entities from lung cancer images.

3.2 Traditional Machine Learning Models

Our analysis involved traditional machine learning models, including Logistic Regression, Decision Tree, Random Forest, Support Vector Machine, and a Dense Neural Network. To prepare the data for these models, pre-processing steps were applied, involving the flattening of images and the implementation of Principal Component Analysis (PCA) to systematically reduce dimensionality. This systematic approach not only optimized the input data for the models but also aimed to enhance their performance by capturing essential features while minimizing computational complexity. The subsequent plot provides a projection of the images into 2D space.

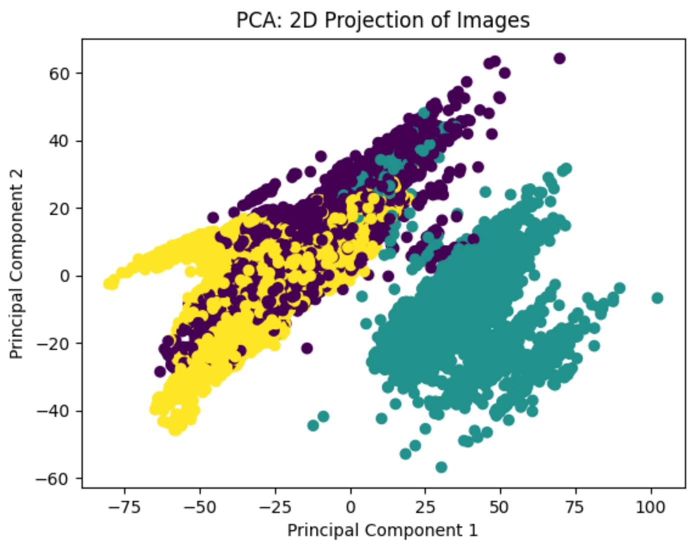
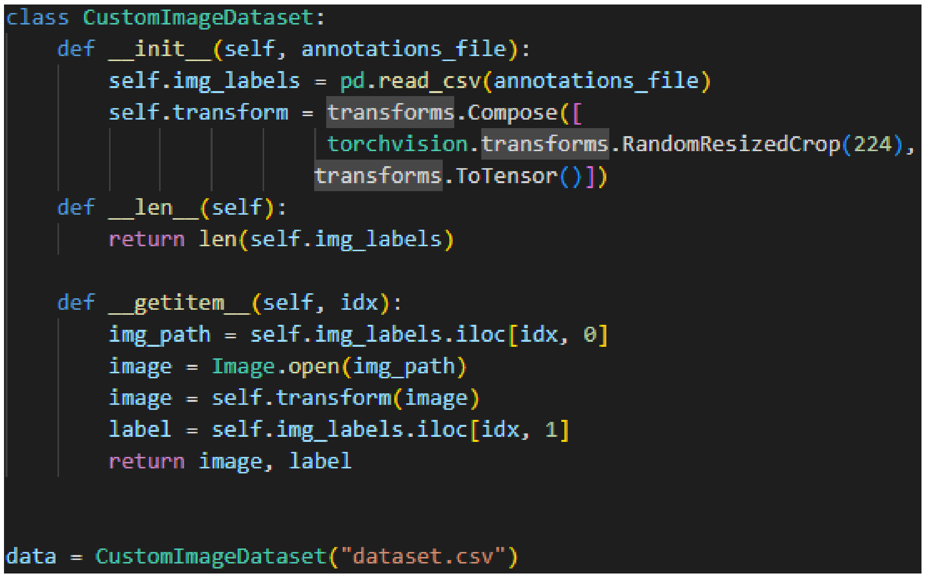


Figure 4: The PCA plot displays a projection of the images into 2D space. (50 principal components were used.) The green class corresponding to Adenocar-

cinoma whereas the yellow and purple correspond to the Benign Tissue and Squamous Cell Carcinoma.

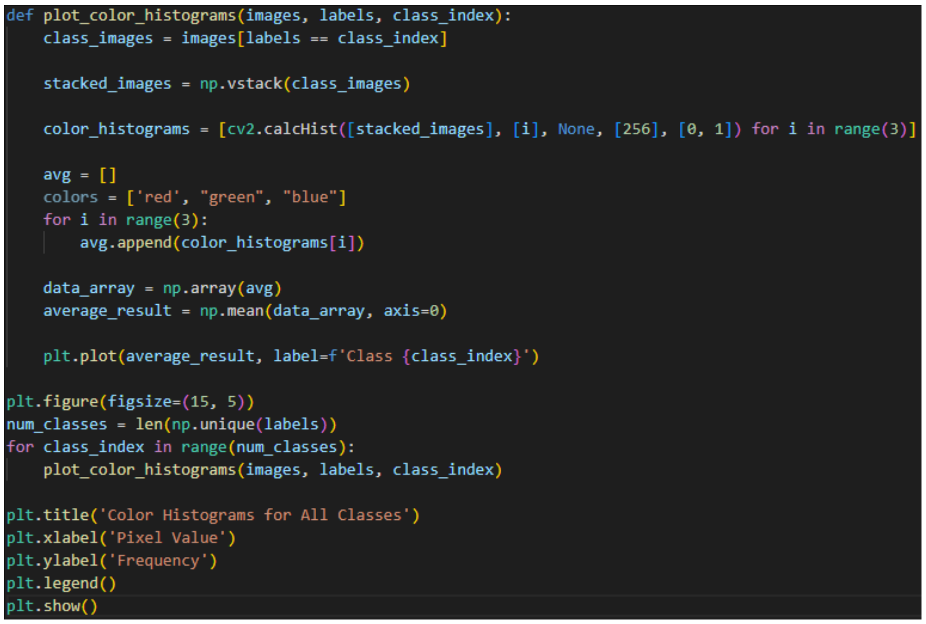
3.2.1 Implementation of Traditional Machine Learning Models

The implementation of traditional machine learning models involved a series of steps utilizing Python, PyTorch [4], and scikit-learn [5] libraries. The process began with a the implementation of a custom dataloader: CustomImageDataset. The function follows Pytorch's dataloader documentation [6] using a CSV file input to load images and labels while also applying transformations, such as resizing and tensor conversion. The output returns image tensors along with corresponding labels.



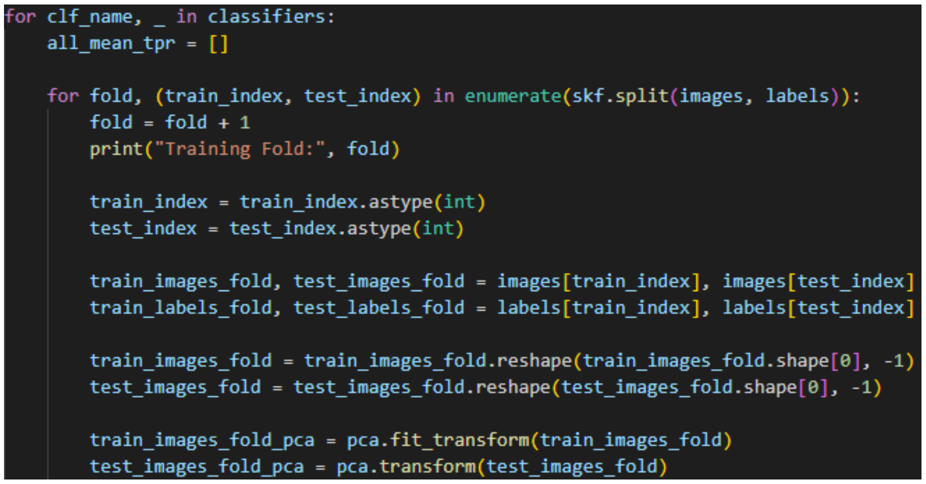
```
class CustomImageDataset:
    def __init__(self, annotations_file):
        self.img_labels = pd.read_csv(annotations_file)
        self.transform = transforms.Compose([
                         torchvision.transforms.RandomResizedCrop(224),
                        transforms.ToTensor()])
    def __len__(self):
        return len(self.img_labels)

    def __getitem__(self, idx):
        img_path = self.img_labels.iloc[idx, 0]
        image = Image.open(img_path)
        image = self.transform(image)
        label = self.img_labels.iloc[idx, 1]
        return image, label


data = CustomImageDataset("dataset.csv")
```

The subsequent code focused on data visualization techniques. Initially, color histograms for different image classes were generated using the OpenCV library, offering insights into pixel value distributions. Dimensionality reduction techniques, including Principal Component Analysis (PCA) and t-distributed Stochastic Neighbor Embedding (t-SNE), were then applied to visualize high-dimensional image data in 2D space.



```
def plot_color_histograms(images, labels, class_index):
    class_images = images[labels == class_index]

    stacked_images = np.vstack(class_images)

    color_histograms = [cv2.calcHist([stacked_images], [i], None, [256], [0, 1]) for i in range(3)]

    avg = []
    colors = ['red', "green", "blue"]
    for i in range(3):
        avg.append(color_histograms[i])

    data_array = np.array(avg)
    average_result = np.mean(data_array, axis=0)

    plt.plot(average_result, label=f'Class {class_index}')

plt.figure(figsize=(15, 5))
num_classes = len(np.unique(labels))
for class_index in range(num_classes):
    plot_color_histograms(images, labels, class_index)

plt.title('Color Histograms for All Classes')
plt.xlabel('Pixel Value')
plt.ylabel('Frequency')
plt.legend()
plt.show()
```

The remaining code is for model training and evaluation. Scikit-learn was employed for various machine learning classifiers. A 5-fold stratified cross-validation was conducted and ROC curves and AUC values were computed to assess the models' discriminatory performance. Mean ROC curves across folds were plotted for visualization.



```
for clf_name, _ in classifiers:
    all_mean_tpr = []

    for fold, (train_index, test_index) in enumerate(skf.split(images, labels)):
        fold = fold + 1
        print("Training Fold:", fold)

        train_index = train_index.astype(int)
        test_index = test_index.astype(int)

        train_images_fold, test_images_fold = images[train_index], images[test_index]
        train_labels_fold, test_labels_fold = labels[train_index], labels[test_index]

        train_images_fold = train_images_fold.reshape(train_images_fold.shape[0], -1)
        test_images_fold = test_images_fold.reshape(test_images_fold.shape[0], -1)

        train_images_fold_pca = pca.fit_transform(train_images_fold)
        test_images_fold_pca = pca.transform(test_images_fold)
```

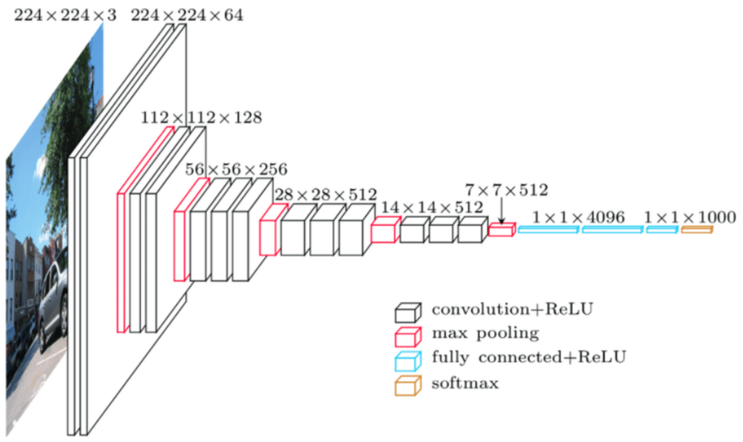
The Support Vector Classifier (SVC) utilized a radial basis function (RBF) kernel with a regularization parameter (C) set to 1.0 and a fixed random seed (random state=1). Logistic Regression was configured with a regularization strength (C) of 1.0 and the same random seed. The Decision Tree Classifier was fitted with a fixed random seed (random state=1). Similarly, the Random Forest Classifier incorporated the same random seed (random state=1). Lastly, the Multi-layer Perceptron (MLP) Classifier featured a single hidden layer comprising 100 neurons, a maximum of 500 iterations, and a consistent random seed (random state=1).

3.3 Deep Learning Models

In recent years, deep learning models have emerged as powerful tools for their ability to automatically learn hierarchical representations and intricate features from complex data. In our investigation, we delved into two prominent deep learning architectures: VGG16 and ResNet18.

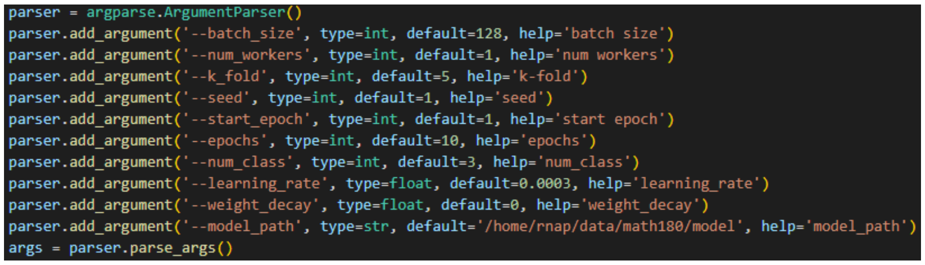
3.3.1 VGG16 Model

The VGG16 model [2], short for Visual Geometry Group 16, is a convolutional neural network (CNN) renowned for its depth and simplicity. Developed by the Visual Geometry Group at the University of Oxford, VGG16 comprises 16 layers, 13 of them being convolutional layers and 3 of them being fully connected layers. One of its distinctive features is the use of small 3x3 convolutional filters throughout the network, combined with max-pooling layers to progressively reduce spatial dimensions. This architecture has proven highly effective in image classification tasks, earning recognition for its straightforward design and impressive accuracy.



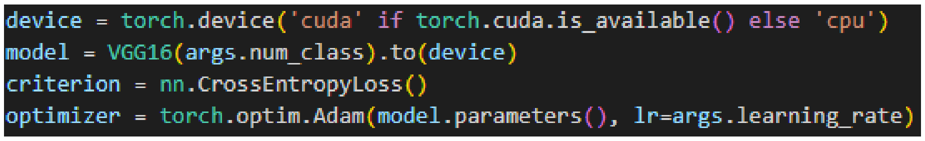
3.3.2 Implementation Details VGG16 Model

The model training process took place on a remote computer, and to make the code adaptable, we started with argument parsing, allowing easy adjustments to parameters using command line arguments. Similar to the traditional machine learning approach, we used the same CustomImageDataset function to load our medical image data.



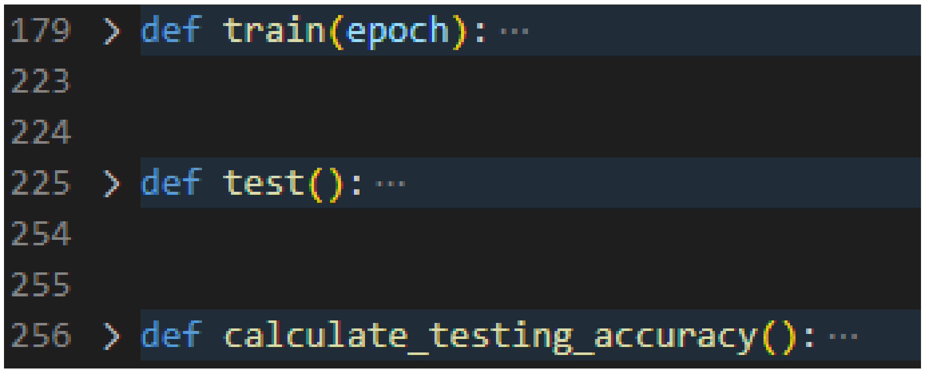
```
parser = argparse.ArgumentParser()
parser.add_argument('--batch_size', type=int, default=128, help='batch size')
parser.add_argument('--num_workers', type=int, default=1, help='num workers')
parser.add_argument('--k_fold', type=int, default=5, help='k-fold')
parser.add_argument('--seed', type=int, default=1, help='seed')
parser.add_argument('--start_epoch', type=int, default=1, help='start epoch')
parser.add_argument('--epochs', type=int, default=10, help='epochs')
parser.add_argument('--num_class', type=int, default=3, help='num_class')
parser.add_argument('--learning_rate', type=float, default=0.0003, help='learning_rate')
parser.add_argument('--weight_decay', type=float, default=0, help='weight_decay')
parser.add_argument('--model_path', type=str, default='/home/rnap/data/math180/model', help='model_path')
args = parser.parse_args()
```

Next, we defined our deep learning model, VGG16 [2], and opted for Adam as the optimization algorithm. For training, we chose CrossEntropyLoss with a learning rate of 0.003, a batch size of 128, and ran for 10 epochs per fold.



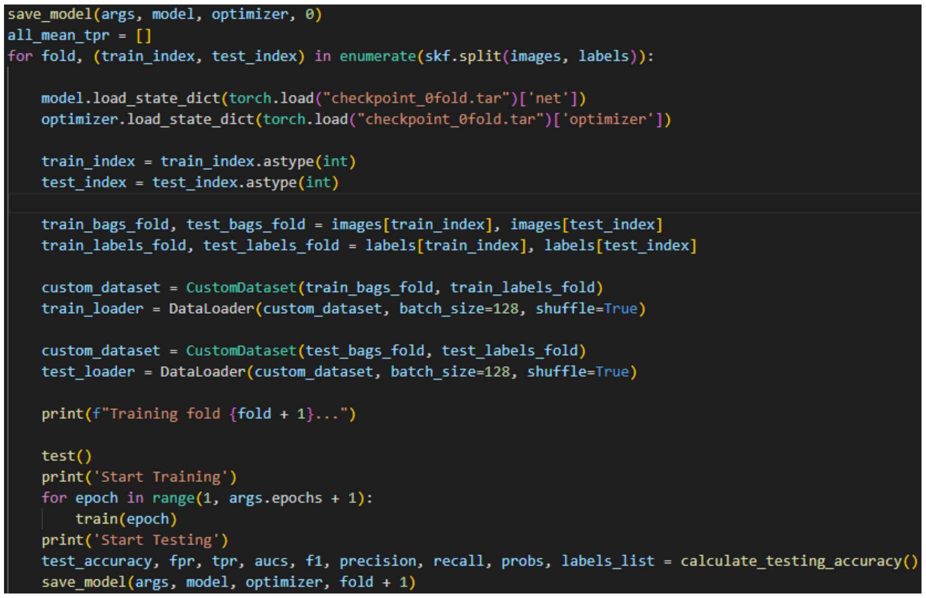
```
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
model = VGG16(args.num_class).to(device)
criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model.parameters(), lr=args.learning_rate)
```

The code then kicks off the training and testing phases, divided into three functions. The first, the train function which updates the model parameters using the training data. Next, the test function performs model inference after each epoch, and the testing assessment function evaluates the model after completing training on each fold. It collects essential information like True Positive Rate (TPR), False Positive Rate (FPR), Area Under the Curve (AUC), and overall accuracies.



```
> def train(epoch): ···


> def test(): ···


> def calculate_testing_accuracy(): ···
```

Finally, the remaining code employs the gathered information to construct an Area Under the Receiver Operating Characteristic (AUROC) plot. This visual representation provides insights into the model's performance across different classes. Additionally, the code calculates averages and standard deviations of accuracies across folds, offering a comprehensive overview of the model's diagnostic capabilities.



```python
save_model(args, model, optimizer, 0)
all_mean_tpr = []
for fold, (train_index, test_index) in enumerate(skf.split(images, labels)):

    model.load_state_dict(torch.load("checkpoint_0fold.tar")['net'])
    optimizer.load_state_dict(torch.load("checkpoint_0fold.tar")['optimizer'])

    train_index = train_index.astype(int)
    test_index = test_index.astype(int)

    train_bags_fold, test_bags_fold = images[train_index], images[test_index]
    train_labels_fold, test_labels_fold = labels[train_index], labels[test_index]

    custom_dataset = CustomDataset(train_bags_fold, train_labels_fold)
    train_loader = DataLoader(custom_dataset, batch_size=128, shuffle=True)

    custom_dataset = CustomDataset(test_bags_fold, test_labels_fold)
    test_loader = DataLoader(custom_dataset, batch_size=128, shuffle=True)

    print(f"Training fold {fold + 1}...")

    test()
    print('Start Training')
    for epoch in range(1, args.epochs + 1):
        train(epoch)
    print('Start Testing')
    test_accuracy, fpr, tpr, aucs, f1, precision, recall, probs, labels_list = calculate_testing_accuracy()
    save_model(args, model, optimizer, fold + 1)
```

The training process was computationally expensive, even with GPU acceleration. Despite utilizing a GPU, the training still took approximately 15 hours to complete. This extended duration can be attributed to the use of 5-fold cross-validation, averaging about 3 hours per fold.

3.3.3 ResNet18 Model

ResNet18 [3], short for Residual Network with 18 layers, represents a significant advancement in deep learning architectures, specifically designed to address the challenge of training very deep networks. Developed by Microsoft Research, ResNet18 incorporates a residual learning framework, allowing input signals to pass through the network without introducing excessive noise. This architecture mitigates the vanishing gradient problem, enabling the training of more profound networks with improved accuracy. The key innovation in ResNet18 lies in its skip connections, or residual connections, which facilitate the flow of information through the network. By alleviating the degradation problem associated with training deep networks, ResNet18 has demonstrated superior performance in various image classification tasks.

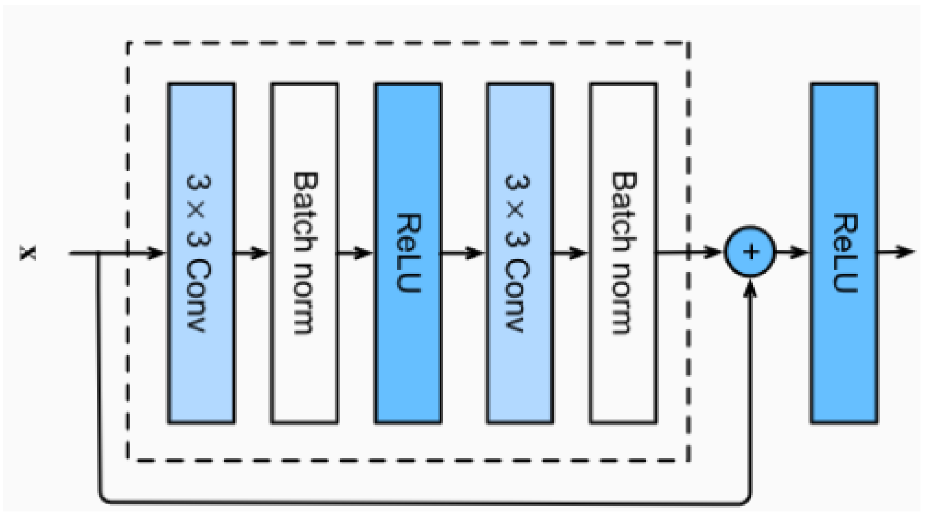
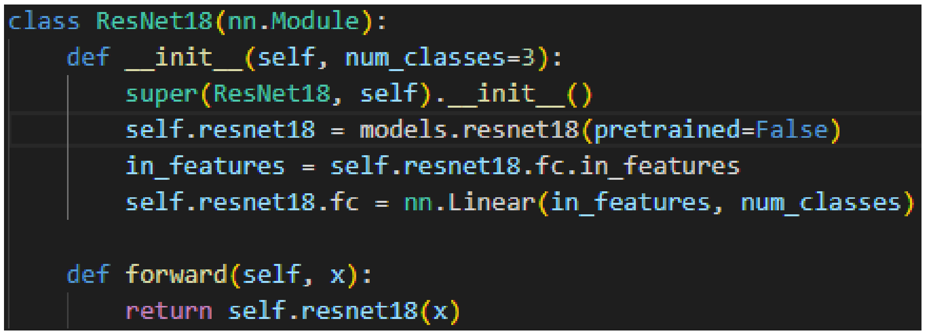


Figure 8: Rather than directly learning the mapping $M(x, \theta)$, ResNet proposes learning the residual $Output - x$, where the output is $x + M(x, \theta)$

3.3.4 Implementation Details ResNet18 Model

The implementation details for the ResNet18 model [3] closely mirror those of the VGG16 model, differing primarily in the architecture of the model. The PyTorch library's built-in ResNet18 model is utilized due to time limitations, without incorporating any pre-trained parameters.



```
class ResNet18(nn.Module):
    def __init__(self, num_classes=3):
        super(ResNet18, self).__init__()
        self.resnet18 = models.resnet18(pretrained=False)
        in_features = self.resnet18.fc.in_features
        self.resnet18.fc = nn.Linear(in_features, num_classes)

    def forward(self, x):
        return self.resnet18(x)
```

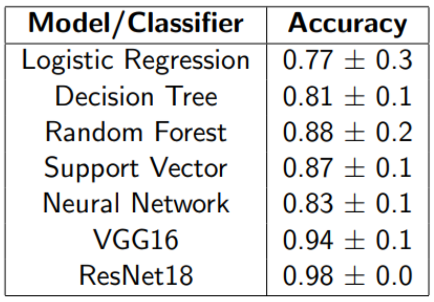
Due to several architectural innovations ResNet18 is much more efficient compared to VGG16, only containing approximately 10 million parameters vs 145 million for VGG16. This accounted to training time of rough 30 minutes.

4 Results

Our findings reveal that traditional machine learning models achieved commendable accuracy, likely attributed to the distinctiveness of the dataset among the three classes. Our preliminary data exploration, particularly through color histograms (Figure 3), demonstrated that the two classes shared a similar color distribution, while the third class exhibited a distinct pattern. This observation is further emphasized in the PCA plot (Figure 4), where two clear clusters are visible. This suggests that distinguishing between the two clusters with similar distributions requires machine learning techniques. While it might not be challenging to classify the two distinct clusters, the presence of three clusters necessitates the application of machine learning to effectively separate the clusters with similar distributions. These findings exemplify the significance of employing both traditional machine learning and deep learning models, to assess whether the increased complexity associated with deep learning models is justified in such situations.

4.1 Machine Learning Model Results

The results of the Logistic Regression Model were the poorest among all classifiers. This suggests that the dataset may not have linear patterns, and the simplicity of Logistic Regression was insufficient for capturing these relationships. On the other hand, the Decision Tree's had a slightly better performance indicating its ability to capture more complex, non-linear relationships within the data. Its likely the model benefited from the hierarchical nature of the decision-making process, allowing it to discern intricate patterns. Of all the traditional classifiers, Random Forest was able to achieve the greatest accuracy showcasing the power of ensemble learning. The Support Vector Classifier demonstrated competitive performance, suggesting that the dataset may have complex decision boundaries. The Dense Neural Network achieved accuracy lower than expected but its performance might be affected by the complexity of the architecture and the need for more data. Overall, it seems Random Forest stands out as a robust performer, but further tuning and exploration could enhance the performance of other models, especially the neural network, by leveraging more sophisticated architectures or fine-tuning hyperparameters.



Model/Classifier	Accuracy
Logistic Regression	0.77 $\pm$ 0.3
Decision Tree	0.81 $\pm$ 0.1
Random Forest	0.88 $\pm$ 0.2
Support Vector	0.87 $\pm$ 0.1
Neural Network	0.83 $\pm$ 0.1
VGG16	0.94 $\pm$ 0.1
ResNet18	0.98 $\pm$ 0.0

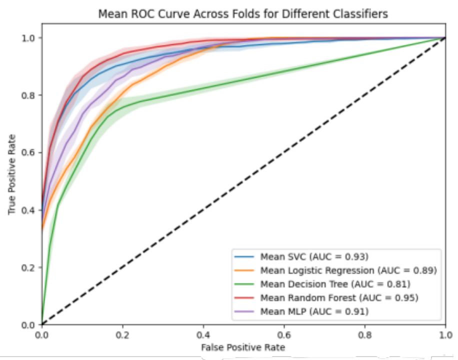


Figure 5: Table of the average accuracy $\pm$ standard deviation across each fold. The AUROC plot uses the lung benign tissue as the dominant class in the one-vs-all approach

4.2 Deep Learning Model Results

Unlike traditional machine learning models that operate on flattened vectors, which often lead to the loss of critical spatial information, the Deep Learning Model utilizes Convolutional Neural Networks (CNNs). This strategic shift gains particular significance in the realm of classifying medical images, where the preservation of spatial details holds importance. Unlike flattened vectors, CNNs can effectively capture intricate patterns and spatial relationships within images, making them well-suited for medical image analysis.

The VGG16 model demonstrated an impressive accuracy of 0.94, as illustrated in Figure 5. However, the inherent depth of deep convolutional networks allows for overfitting to the training data. To address this, it became essential to monitor and evaluate testing accuracy across epochs, as depicted in Figure 8. While a 0.94 accuracy is noteworthy, the pursuit of improvement sparked a deeper investigation to identify research addressing the challenges observed.

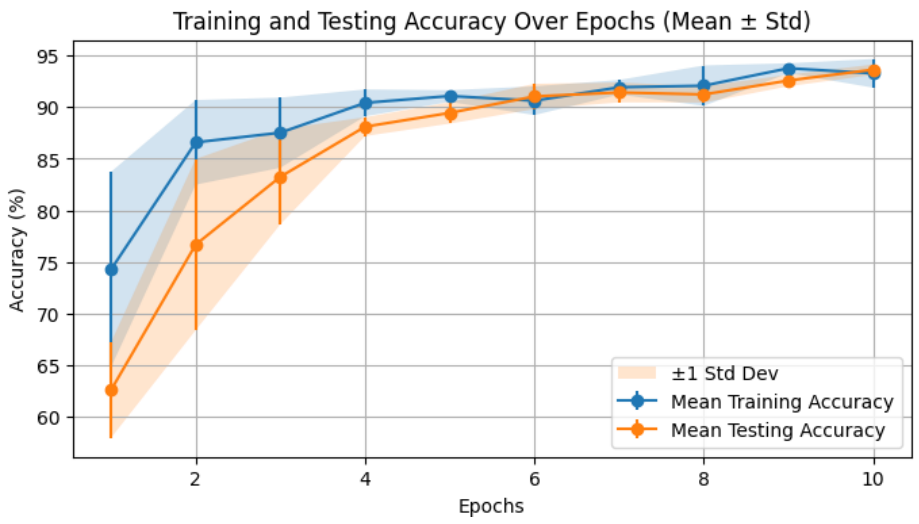


Figure 8: Here is the training and testing accuracies over epoch across folds for VGG16

Training deep neural networks poses a significant challenge due to the multitude of randomized parameters within the network. When an input x is processed through the network, it undergoes multiplication by these randomized parameters, introducing random noise that can overshadow the actual input signal. This phenomenon makes training deep neural networks challenging. However, in 2015, researchers from Microsoft introduced a groundbreaking concept: residual learning [3]. Instead of directly learning the mapping $M(x, \theta)$, they proposed learning the residual $Output - x$, where the output is $x + M(x, \theta)$, as depicted in Figure 7. Surprisingly, this approach proved to be more manageable, allowing for effective training of deep neural networks. Applying this

ResNet architecture, we achieved a remarkable performance of 0.98 accuracy, showcasing the effectiveness of the ideas presented in the paper.

4.3 Limitations

While our study has yielded promising results, it is essential to acknowledge its inherent limitations. The performance of our model may be susceptible to external factors that were not considered in our analysis. The relatively small size of the dataset could potentially limit the model's ability to comprehensively capture the diverse spectrum of variations present in medical images. Moreover, computational constraints compelled us to apply Principal Component Analysis (PCA) only to the machine learning methods, not to the deep learning models, potentially introducing a discrepancy in the comparison. Additionally, the use of stratified splitting in our dataset may not entirely mirror real-world scenarios, as it assumes an equal distribution of cancerous and noncancerous tissues, which might not be the case. Furthermore, the augmentation process applied by the dataset authors raises concerns, as starting with 250 images and augmenting them to 5000 images per class may impact the diversity and authenticity of the dataset. These limitations should be considered when interpreting and generalizing the outcomes of our study.

4.4 Future Directions

Given more time and resources, there are several promising avenues for further exploration and refinement of our lung cancer diagnosis framework.

Transfer Learning for Improved Efficiency

One potential direction involves leveraging the lung cancer dataset for transfer learning. The application of pre-trained models on a related datasets could reduce the need for computationally extensive training of deep convolutional neural networks. Transfer learning allows the model to leverage knowledge gained from one task (e.g., lung cancer detection) to improve performance on a related but different task (e.g., colon cancer detection).

Exploring Image Preprocessing Impact

Exploring the impact of image preprocessing, such as converting all images to black and white, is another avenue worth exploring. This could potentially highlight whether certain color information is essential for accurate lung cancer diagnosis or if a simplified representation still maintains high performance. Understanding the sensitivity of the models to color information could provide valuable insights into the features that contribute most to the diagnostic process.

Nuanced Evaluation with Extended Metrics

While our study focused on overall accuracy, incorporating additional performance metrics such as F1 score and confusion matrices could provide a more nuanced evaluation of model performance. These metrics offer insights into the model's ability to correctly identify positive cases (sensitivity) and negative cases (specificity), as well as its performance in handling false positives and false negatives.

Fine-Tuning Traditional Models for Optimization

Further investigation into fine-tuning traditional machine learning models is also warranted. Optimizing hyperparameters and exploring feature engineering techniques may unlock additional potential in these models. This comparative analysis would contribute valuable insights into the strengths and weaknesses of both traditional and deep learning approaches.

5 Conclusion

Our comprehensive investigation into lung cancer diagnosis using machine learning and computer vision techniques yielded noteworthy outcomes. While traditional machine learning models exhibited reasonable accuracy, the real breakthrough materialized with the deployment of deep convolutional neural networks, showcasing their pivotal role in shaping the development of robust clinical decision support systems. These advanced networks offer exciting prospects for revolutionizing the accuracy and speed of lung cancer diagnosis, marking a new era in medical imaging technology. The transformative potential is evident; if machine learning consistently attains high accuracies in these tasks, automated systems can expedite treatment decisions, elevate patient outcomes, and alleviate the burden on healthcare professionals. The integration of machine learning in lung cancer diagnosis holds the promise of enhancing efficiency and precision in clinical practice, representing a significant advancement in medical imaging technology. Nevertheless, we acknowledge limitations with medical imaging related to data availability and computational resources for the ongoing exploration and refinement in this evolving field. Supplementary code associated with this report can be found at:
`https://github.com/napronald/MATH180/tree/main/Project`

6 Acknowledgements

We acknowledge the Lung and Colon Cancer Histopathological Image Dataset (LC25000) [1] and the AI researchers contributing to cancer pathology. Their efforts have been instrumental in advancing research in this critical domain.

References

[1] Lung and Colon Cancer Histopathological Image Dataset (LC25000). Available at: `https://arxiv.org/ftp/arxiv/papers/1912/1912.12142.pdf`

[2] Simonyan, K., & Zisserman, A. (2014). Very Deep Convolutional Networks for Large-Scale Image Recognition. *arXiv preprint arXiv:1409.1556*. Retrieved from `https://arxiv.org/pdf/1409.1556.pdf`

[3] He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep Residual Learning for Image Recognition. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Retrieved from `https://arxiv.org/pdf/1512.03385.pdf`

[4] Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., ... Lerer, A. (2019). PyTorch: An Imperative Style, High-Performance Deep Learning Library. *Advances in Neural Information Processing Systems*, 32. Retrieved from `https://arxiv.org/abs/1912.01703`

[5] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., ... Vanderplas, J. (2011). Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, 12, 2825-2830. Retrieved from `https://scikit-learn.org/stable/about.html#citing-scikit-learn`

[6] PyTorch. (n.d.). Data Loading and Processing Tutorial. Retrieved from `https://pytorch.org/tutorials/beginner/basics/data_tutorial.html`