

Histopathologic Cancer Detection

Chandan(M22PH006)^{1*}

Tushar(M22ME058)²

Aditya(M22MT006)³

November 25, 2023

Abstract

Our project aims to develop a histopathologic cancer detection system utilizing deep learning techniques. We utilized the PCam dataset, which contains a vast collection of small pathology images, for classification purposes. The images are labeled, with positive labels indicating the presence of tumor tissue in the central 32×32 px region. The outer region of the patches does not influence the label, facilitating consistent behavior for fully-convolutional models without zero-padding.

Our approach involved data preprocessing, exploratory data analysis, and the implementation of a Convolutional Neural Network (CNN) using transfer learning with a pre-trained ResNet model. We performed model training and evaluation using metrics like accuracy, precision, recall, F1 score, and confusion matrix to assess its performance in cancer detection.

Contents

Introduction

Histopathologic cancer detection is a critical task in medical diagnostics, where the identification of tumor tissue in pathology images aids in diagnosis and treatment planning. Our project focuses on leveraging machine learning techniques to automate this process, potentially assisting medical professionals in accurate and timely cancer identification.

Dataset and Challenge

We utilized the PCam dataset, which consists of pathology images labeled with information regarding tumor presence within the central region. This dataset posed a classification challenge, particularly in distinguishing tumor tissue within the specified area.

Methodology

Our methodology involved various stages:

- **Data Preparation:**

- **Data Preparation I**

I'm initializing crucial libraries like pandas, OpenCV, and PyTorch for data handling, image processing, and machine learning. Setting paths for training and testing datasets, I load CSV files that contain labels and merge them with image file paths, creating a unified dataframe. Using OS functions, I retrieve image files from a particular directory, generate unique identifiers for each image, and combine them with respective labels. The resulting dataframe displays the first ten entries, enabling an initial inspection of the merged data before subsequent steps like model training and evaluation.

```
1 #!/usr/bin/env python
2 # coding: utf-8
3
4 # In[ ]:
```

*Author One was partially supported by Grant XXX

```

5
6
7 import os
8 import pandas as pd
9 import cv2
10 import matplotlib.pyplot as plt
11 from sklearn.model_selection import train_test_split as tts
12 from sklearn.metrics import accuracy_score, precision_score,
    recall_score, f1_score, confusion_matrix
13
14 import torch
15 import torch.nn as nn
16 import torch.nn.functional as F
17 import torchvision
18 import torchvision.transforms as transforms
19 from torch.utils.data import TensorDataset, DataLoader, Dataset
20
21 data_train_path = "D:\\IIT JODHPUR\\JUPYTER WORK\\FML\\Class lab\\
    PROJECT\\train"
22 data_test_path = "D:\\IIT JODHPUR\\JUPYTER WORK\\FML\\Class lab\\
    PROJECT\\test"
23 data_train_labels = pd.read_csv("D:\\IIT JODHPUR\\JUPYTER WORK\\FML
    \\Class lab\\PROJECT\\train_labels\\train_labels.csv")
24
25 ### Creating the dataframe and merging the labels from the folder
26
27 tif_files = [os.path.join(data_train_path, filename) for filename in
    os.listdir(data_train_path)
28               if filename.endswith('.tif')]
29 df = pd.DataFrame({'path': tif_files})
30 #print(df.head())
31 #df['path'][0]
32 df['id'] = df['path'].map(lambda x: x.split('\\')[7].split(".")[0])
33 #df['id'][0]
34 #df.head(5)
35 df = df.merge(data_train_labels, on = "id") # merged labels and
    filepaths
36 df.head(10)

```

	path	id	label
0	D:\IIT JODHPUR\JUPYTER WORK\FML\Class lab\PROJ...	00001b2b5609af42ab0ab276dd4cd41c3e7745b5	1
1	D:\IIT JODHPUR\JUPYTER WORK\FML\Class lab\PROJ...	000020de2aa6193f4c160e398a8edea95b1da598	0
2	D:\IIT JODHPUR\JUPYTER WORK\FML\Class lab\PROJ...	00004aab08381d25d315384d646f5ce413ea24b1	0
3	D:\IIT JODHPUR\JUPYTER WORK\FML\Class lab\PROJ...	0000d563d5cfafe68ac7c9829258a298d9b6a	0
4	D:\IIT JODHPUR\JUPYTER WORK\FML\Class lab\PROJ...	0000da768d06b879e5754c43e2298ce48726f722	1
5	D:\IIT JODHPUR\JUPYTER WORK\FML\Class lab\PROJ...	0000f8a4da4c286eee5cf1b0d2ab82f979989f7b	0
6	D:\IIT JODHPUR\JUPYTER WORK\FML\Class lab\PROJ...	00010f78ea8f878117500c445a658e5857f4e304	0
7	D:\IIT JODHPUR\JUPYTER WORK\FML\Class lab\PROJ...	00011545a495817817c6943583b294c900a137b8	0
8	D:\IIT JODHPUR\JUPYTER WORK\FML\Class lab\PROJ...	000126ec42770c7568204e2f6e07eb9a07d5e121	0
9	D:\IIT JODHPUR\JUPYTER WORK\FML\Class lab\PROJ...	00014e39b5df5f80df56f18a0a049d1cc6de430a	1

Figure 1: New Dataframe of file path

– Data Preparation II :

Due to our limited computational capacity, we had to trim down the files to 50,000 images. I created a function called `load_data` to handle loading N images from the given dataframe 'df.' Each image is resized to a size of 96x96 pixels and possesses 3 channels, indicating an RGB format. This function also retrieves the image labels from the dataframe and stores them in 'y.' The images are loaded into 'X' using OpenCV. To ensure consistent values, I normalized the pixel values to range between 0 and 1. Additionally, I generated eight random samples paired with their respective labels and displayed them using Matplotlib. This visualization helps in grasping a glimpse of the images and understanding their corresponding labels.

```

1  #!/usr/bin/env python
2  # coding: utf-8
3
4  # In[ ]:
5
6
7  # Due to limited computational power, we had to cut down files to
   50,000 images.
8
9  def load_data(N, df):
10     """ This function loads N images using the data df """
11     X = np.zeros([N, 96, 96, 3], dtype=np.uint8)      # 3
12     inputs suggest the files are of RGB type
13     y = np.squeeze(df['label'].values)[:N]
14
15     for i in range(N):
16         X[i] = cv2.imread(df['path'][i])
17
18     return X, y
19
20 N = 50000
21 x,y = load_data(N,df)
22
23 Normalization for values to lie under 0-1 scale
24
25 x = x / 255.0 # maximum pixel values
26
27 Generating 8 random samples with labels and plotting them
28
29 fig = plt.figure(figsize=(8, 9), dpi=150)
30 np.random.seed(100)
31 image_count = 8 # number of images we want to have
32
33 for plot_idx, image_idx in enumerate(np.random.randint(0, N,
34     image_count)):
35     ax = fig.add_subplot(4, 4, plot_idx+1)
36     plt.imshow(x[image_idx])
37     ax.set_title("Label: " + str(y[image_idx]))

```

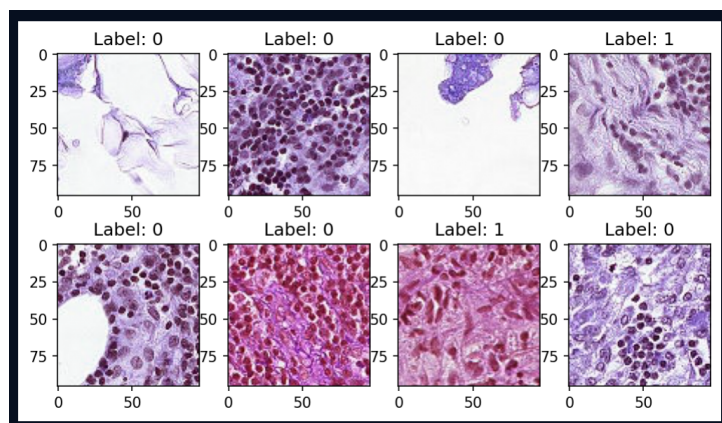


Figure 2: sample images with labels

1. Exploratory Data Analysis (EDA):

– Exploratory Data Analysis I:

I plotted a count plot to assess the balance of the data after downsizing the files. I created a figure using Matplotlib with a specific size and dpi for visualization. Calculating the

count of negative and positive instances in the labels helped create separate arrays for negative and positive image samples. By using a bar plot, I displayed the counts of negative and positive samples on the y-axis against their respective labels on the x-axis, providing a visual comparison. This allowed a quick evaluation of the dataset's class distribution post the reduction in files.

```

1 #!/usr/bin/env python
2 # coding: utf-8
3
4 # In[ ]:
5
6
7 # Checking the count plot to check whether the data is balanced or
  not since we had to cut down few files
8
9 fig = plt.figure(figsize=(4, 2),dpi=100)
10 number_of_negatives = (y==0).sum()
11 number_of_positives = (y==1).sum()
12 negative_samples = x[y==0]
13 positive_samples = x[y==1]
14 plt.bar([1, 0], [number_of_negatives, number_of_positives])
15 plt.xticks([1, 0], ["Negative N = " + str(number_of_negatives),
16                     "Positive N = " + str(number_of_positives)])
17 plt.ylabel("# of samples");

```

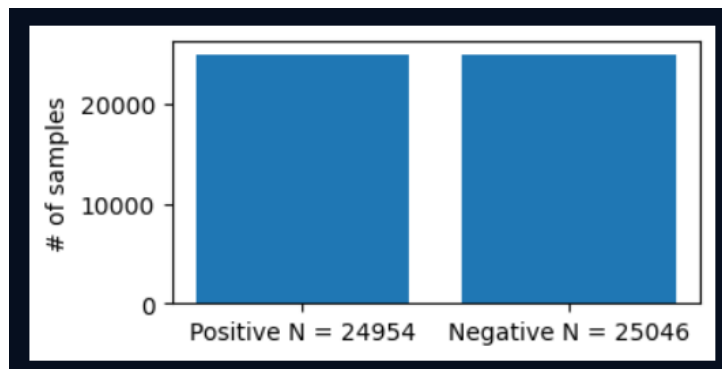


Figure 3: bar plot of number of samples

– Exploratory Data Analysis II:

In this segment of code, I utilized Matplotlib to generate subplots displaying histograms of pixel values across Red, Green, and Blue color channels, as well as the cumulative RGB for both positive and negative samples. The 'nr_of_bins' variable was set to 256, determining the number of bins for the histograms. Using a nested loop structure, I iterated through rows and columns in the subplot grid, setting labels and titles for better visualization. The histograms depicted the relative frequency of pixel intensities in each color channel for positive and negative samples, aiding in understanding the distribution of pixel values. Additionally, by comparing these histograms, distinct patterns between positive and negative samples in the dataset could be discerned, contributing to the analysis of pixel intensity distributions.

```

1 #!/usr/bin/env python
2 # coding: utf-8
3
4 # In[ ]:
5
6
7 ''' We plot subplots to visualize histograms of pixel values in
  image color channels (Red, Green, Blue)
8 and cumulative RGB for both positive and negative samples. This
  analysis aids in understanding the

```

```

9 distribution of pixel intensities across color channels and
10 distinguishes patterns between positive
11 and negative samples in the dataset. '''
12
13 nr_of_bins = 256
14 fig, axes = plt.subplots(4, 2, sharey=True, figsize=(8, 8), dpi=120)
15 rgb_list = ["Red", "Green", "Blue", "RGB"]
16
17 for row_idx in range(0, 4):
18     for col_idx in range(0, 2):
19         if row_idx < 3:
20             axes[row_idx, 0].set_ylabel("Relative Frequency")
21             axes[row_idx, 1].set_ylabel(rgb_list[row_idx], rotation="
22                                     horizontal",
23                                     labelpad=35, fontsize=12)
24
25             if col_idx == 0:
26                 axes[row_idx, 0].hist(positive_samples[:, :, :,
27                 row_idx].flatten(),
28                                     bins=nr_of_bins, density = True
29                                     )
30             elif col_idx == 1:
31                 axes[row_idx, 1].hist(negative_samples[:, :, :,
32                 row_idx].flatten(),
33                                     bins=nr_of_bins, density = True
34                                     )
35
36         else:
37             # rgb as cumulative
38             if col_idx == 0:
39                 axes[row_idx, 0].hist(positive_samples.flatten(),
40                                     bins=nr_of_bins, density = True
41                                     )
42             elif col_idx == 1:
43                 axes[row_idx, 1].hist(negative_samples.flatten(),
44                                     bins=nr_of_bins, density = True
45                                     )
46
47 axes[0, 0].set_title("Positive Samples")
48 axes[0, 1].set_title("Negative Samples")

```

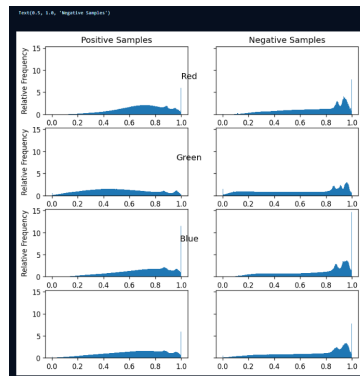
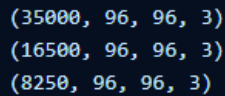


Figure 4: relative frequency of pixel intensities for the two classes

2. Data Splitting:

I partitioned the data into training, testing, and validation subsets to facilitate machine learning model development and evaluation. Utilizing the `train_test_split` function from `sklearn`, I divided the dataset x and corresponding labels y into a training set (70%) and a temporary set (30%). Further division of the temporary set resulted in the creation of validation (33% of the original data) and test sets (67% of the temporary set). This stratification was employed to maintain the class distribution within each subset. Displaying the shapes of the training, validation, and test data subsets provided insights into their respective sizes, aiding in understanding the dataset's partitioning.

```
1 #!/usr/bin/env python
2 # coding: utf-8
3
4 # In[ ]:
5
6
7 # Splitted the data for training , testing and validation purpose
8
9 x_train,x_temp,y_train,y_temp = tts(x,y,test_size = 0.3 ,shuffle =True
10 ,stratify=y)
11 print(x_train.shape)
12 x_val,x_test,y_val,y_test = tts(x_temp,y_temp,test_size = 0.33 ,shuffle
13 =True ,stratify=y_temp)
14 print(x_test.shape)
15 print(x_val.shape)
```



```
(35000, 96, 96, 3)
(16500, 96, 96, 3)
(8250, 96, 96, 3)
```

Figure 5: shape of training, testing & validation sets

3. Model Implementation:

The ResNet (Residual Network) model, initially trained on the ImageNet dataset, is a deep neural network renowned for its depth and utilization of residual learning blocks. Here, we incorporated ResNet due to its proven efficacy in handling deep architectures while mitigating the vanishing gradient problem. This model architecture introduced residual connections, enabling the network to learn residual mappings instead of directly learning underlying functions. By leveraging a pre-trained ResNet model from `torchvision`, we utilized transfer learning to adapt its already learned features for our histopathologic image classification task. This approach helped us benefit from the ResNet model's robust feature extraction capabilities and significantly improve the accuracy of our classification beyond what was achievable with a custom-built CNN model.

4. Model Training and Evaluation:

The attempt to build our custom CNN model yielded an accuracy around 0.61, prompting us to turn to a ResNet Model for enhanced performance. The ResNet (Residual Network) model was initially trained on the ImageNet dataset, a comprehensive image classification dataset containing over a million labeled images distributed across a thousand classes. By utilizing the `torchvision` library, I accessed the ResNet architecture, which is pretrained on ImageNet. Subsequently, I instantiated our ResNet model, modifying the final fully connected layers to match the desired output size for our classification task. Employing the Cross Entropy Loss criterion and the Adam optimizer from the PyTorch library, I initialized these for model training. During training for 10 epochs, the model underwent the optimization process, adjusting its parameters to minimize the loss on the training data. After training, the model was evaluated using the test dataset to assess its performance and validate its effectiveness in classifying the histopathologic images.

```
1 #!/usr/bin/env python
2 # coding: utf-8
```

```

3
4 # In[ ]:
5
6
7 ### Transfer Learning with pretrained ResNet Model
8
9 '''We had tried to build our own CNN model but that was giving us an
10 accuracy around 0.61.
11 Hence we took help of ResNet Model .The ResNet (Residual Network) model
12 was originally introduced
13 and trained on the ImageNet dataset. The ImageNet dataset is a large-
14 scale dataset for image
15 classification and contains over a million labeled images across a
16 thousand different classes.'''
17
18 import torchvision.models as models
19
20 class ResNetModel(nn.Module):
21     def __init__(self):
22         super(ResNetModel, self).__init__()
23         resnet18 = models.resnet18(pretrained=True) # latest
24         improvements are from 2018
25
26         self.features = nn.Sequential(*list(resnet18.children())[:-1])
27
28         self.fc1 = nn.Linear(512, 128) # Modified final fully connected
29         layers to match desired output size
30         self.fc2 = nn.Linear(128, 1)
31
32     def forward(self, x):
33         x = self.features(x)
34         x = x.view(x.size(0), -1)
35         x = F.relu(self.fc1(x))
36         x = self.fc2(x)
37         return x
38
39 model_cnn = ResNetModel()
40 criterion = nn.CrossEntropyLoss()
41 optimizer = torch.optim.Adam(model_cnn.parameters(), lr=0.001)
42
43 x_train_tensor = torch.Tensor(x_train).permute(0, 3, 1, 2)
44 y_train_tensor = torch.Tensor(y_train.reshape(-1, 1))
45
46 for epoch in range(10):
47     model_cnn.train()
48     optimizer.zero_grad()
49     outputs = model_cnn(x_train_tensor)
50     loss = criterion(outputs, y_train_tensor)
51     loss.backward()
52     optimizer.step()
53
54 model_cnn.eval()
55 x_test_tensor = torch.Tensor(x_test).permute(0, 3, 1, 2)
56 y_test_tensor = torch.Tensor(y_test.reshape(-1, 1))

```

5. Result:

The thresholds, such as greater than 0.5, are employed in this scenario to convert the model's probability predictions into binary outcomes of 0 or 1. Utilizing PyTorch, I applied the trained model, `model_cnn`, to predict outcomes on the test data. By using the `torch.sigmoid` function, I converted the predictions to probabilities and then to binary predictions based on the defined threshold. Subsequently, I computed various performance metrics like accuracy,

precision, recall, F1 score, and the confusion matrix using functions provided by the scikit-learn library. Finally, I printed these metrics to assess the CNN model's performance in classifying histopathologic images, providing a comprehensive evaluation of its effectiveness. It's noteworthy that we observed very few (almost zero) False Positives, indicating promising model performance.

```
1 #!/usr/bin/env python
2 # coding: utf-8
3
4 # In[ ]:
5
6
7 '''The thresholds (e.g., > 0.5) are used here to convert probability
8 predictions into binary predictions
9 0 or 1. '''
10
11 with torch.no_grad():
12     y_pred_cnn = model_cnn(x_test_tensor)
13     y_pred_cnn = torch.sigmoid(y_pred_cnn).numpy()
14
15 accuracy_cnn = accuracy_score(y_test, y_pred_cnn > 0.5)
16 precision_cnn = precision_score(y_test, y_pred_cnn > 0.5)
17 recall_cnn = recall_score(y_test, y_pred_cnn > 0.5)
18 f1_score_cnn = f1_score(y_test, y_pred_cnn > 0.5)
19 confusion_matrix_cnn = confusion_matrix(y_test, y_pred_cnn > 0.5)
20
21 # Printing the accuracy measurements for the classification
22
23 print("CNN Model Metrics:")
24 print("Accuracy:", accuracy_cnn)
25 print("Precision:", precision_cnn)
26 print("Recall:", recall_cnn)
27 print("F1 Score:", f1_score_cnn)
28 print("Confusion Matrix:")
29 print(confusion_matrix_cnn)
```

```
CNN Model Metrics:
Accuracy: 0.96345987357864
Precision: 0.8760683760683761
Recall: 0.21674876847290642
F1 Score: 0.275
Confusion Matrix:
[48173, 0]
[159, 44]
```

Figure 6: Performance of the model