Data Collection and Preprocessing Phase

Date	15 March 2024
Team ID	LTVIP2024TMID24981
Project Title	Deep learning techniques for breast cancer prediction
Maximum Marks	6 Marks

Data Exploration and Preprocessing Report

A **Data Exploration and Preprocessing Report** is a crucial step in developing a robust model for **breast cancer prediction**. The goal is to thoroughly explore the dataset, understand its structure, and apply the necessary preprocessing steps before feeding the data into a machine learning model such as a CNN. Below is an outline of what such a report would include, tailored to a breast cancer prediction project using images (e.g., mammograms or histopathology slides).

Section	Description

Data Overview	Breast cancer prediction using deep learning is an active research area aimed at improving early detection, diagnosis, and prognosis of breast cancer. Deep learning methods, particularly convolutional neural networks (CNNs).
	• **Dataset Source**: Identify the source of the dataset (e.g., publicly available datasets like **Break His**, **DDSM**, or a hospital dataset).
	• **Data Type**: Describe the type of data (e.g., mammograms, histopathology images, or both).
	    ○ **Mammograms**: Typically, grayscale images used to detect masses or calcifications.
	    ○ **Histopathology Slides**: Usually RGB-stained images used to analyse cellular structures.
	• **Dataset Size**: Provide the number of samples, including breakdowns by class (e.g., benign vs. malignant).
	• **Labels**: Outline the labels and their distribution. Common labels might include:
	    ○ **Benign**
	    ○ **Malignant**
	    ○ **Normal**
	1.2. Class Distribution

	• **Class Imbalance**: Explore the distribution of classes (benign, malignant, normal). If there's a significant imbalance (e.g., far more benign than malignant cases), consider strategies like oversampling, under sampling, or synthetic data generation (e.g., SMOTE). **1.3. Image Dimensions** • **Image Sizes**: Check the dimensions of the images. Often, images from different sources may have varying resolutions. This impacts how we process and resize them. • **Colour Channels**: Investigate whether the images are grayscale or RGB. Mammograms are usually grayscale, while histopathology slides are RGB. **1.4. Missing Data** • **Check for Missing Images or Labels**: Ensure that all images have corresponding labels and there are no corrupted files in the dataset. **1.5. Visual Inspection** • **Sample Images**: Display a few random images from each class (benign, malignant) to gain an intuitive understanding of the dataset. This helps in understanding image quality and variability.
Resizing	• **Objective**: Standardize the input image size for the model. • **Why Resize**: Different imaging devices can produce images of varying dimensions. Neural networks typically require all inputs to have a fixed size. For instance, you may resize all images to 224x224 for compatibility with models like Resnet or Efficient Net. • **Consideration**: While resizing, preserving the aspect ratio is crucial to avoid distortion of important features like cell structures or tumour boundaries.
Normalization	• **Objective**: Ensure that the pixel intensity values are on a similar scale for better training stability. • **Why Normalize**: Medical images (like histopathology or mammograms) can have varying intensity ranges due to differences in scanners, staining techniques, or patient conditions. Normalizing the pixel values (scaling to [0, 1] or [-1, 1]) ensures that the learning algorithm treats all features uniformly. • **Techniques**: Common normalization approaches include subtracting the mean and dividing by the

	standard deviation for pixel intensities.
Data Augmentation	**Objective**: Artificially increase the dataset size to make the model more robust. **Why Augment**: Medical imaging datasets are often small and imbalanced. Data augmentation can help mitigate this by applying transformations such as: • **Rotations**: Slightly rotating images can help the model recognize tumours regardless of the orientation. • **Flips**: Horizontal and vertical flips can help the model learn symmetries. • **Zoom**: Zooming in or out simulates variability in zoom levels between different images. • **Contrast Adjustments**: Varying contrast can help simulate different imaging conditions.

Denoising	• **Objective**: Reduce noise to improve image quality and model performance.
	• **Why Denoise**: Medical images, especially histopathological images, can have noise due to artifacts from the imaging process, patient movement, or machine errors.
	• **Techniques**:
	    ○ **Gaussian Blur**: Reduces high-frequency noise, which may not be relevant for feature extraction.
	    ○ **Median Filtering**: Effective for removing 'salt-and-pepper' noise that might appear due to image acquisition errors.
	    ○ **Deep Learning Methods**: Denoising autoencoders or other deep learning-based techniques can be trained to remove noise while preserving important features like tumour boundaries.

Edge Detection	**Objective**: Highlight the boundaries of tumours or other regions of interest. **Why Edge Detection**: Tumour boundaries and irregular shapes are important in cancer prediction models. Edge detection helps isolate these features for analysis. **Techniques**: • **Sobel Filter**: Simple technique for detecting edges based on intensity gradients.

	• **Canny Edge Detection**: More sophisticated, useful for detecting sharp changes in intensity which often correspond to tumour borders. • **Application**: Edge detection can help segment regions of interest (e.g., tumours) from surrounding tissues, improving the focus on important features.
Color Space Conversion	**Objective**: Convert images from one colour space (like RGB) to another (such as grayscale or HSV) to highlight specific features useful for prediction. **Why Convert Colour Spaces**: • **Histopathology Images**: In breast cancer prediction, histopathological images are often stained using techniques like H&E (hematoxylin and eosin). Colour information in these images can be crucial for identifying cancerous tissues. • **Mammograms**: Mammograms are usually grayscale images, and colour space conversion may not be necessary. However, processing them in different intensity ranges can help highlight contrasts in the tissue.

Image Cropping	• **Objective**: Crop out unnecessary parts of the image to focus on the region of interest (ROI), such as a tumour.
	• **Why Crop**: Medical images often contain a lot of background information that may not be relevant for the model. Cropping helps:
	    ○ **Remove Background Noise**: Focus on key areas where tumours or abnormalities are likely to exist.
	    ○ **Reduce Image Size**: Cropping can reduce image dimensions, which speeds up model training and reduces computational load.
	    ○ **Improves Model Performance**: The CNN will focus on the important parts of the image (e.g., suspicious regions in a mammogram or biopsy), leading to better learning and prediction accuracy.

Batch Normalization	**Objective**: Normalize the input features across a mini-batch, improving training speed and model stability. **Why Use Batch Normalization?**: In deep learning, the internal covariate shift (changing data distributions during training) can

<table>
<tr>
<td></td>
<td>
slow down learning. Batch normalization helps by:

Stabilizing Learning: Normalizes activations layer by layer, ensuring that the distribution of inputs to each layer stays consistent throughout training.
Faster Convergence: By maintaining stable gradients, batch normalization enables faster and more efficient training of deep CNNs.
Regularization: It acts as a form of regularization, reducing overfitting by introducing noise to the network's activations.

</td>
</tr>
<tr>
<td colspan="2">Data Preprocessing Code Screenshots</td>
</tr>
<tr>
<td>Loading Data</td>
<td>
This step involves loading the image data and resizing it to a fixed size (e.g., 224x224).

```
#function to load differnet labels found in dataset
path = "Dataset"
labels = []
X = []
Y = []
for root, dirs, directory in os.walk(path):
    for j in range(len(directory)):
        name = os.path.basename(root)
        if name not in labels:
            labels.append(name.strip())
print(labels)
```

```
['benign', 'malignant', 'normal']
```
</td>
</tr>
</table>

Normalization	Scaling the pixel values between 0 and 1.
	`]: #preprocess images like shuffling and normalization`
	`X = X.astype('float32')`
	`X = X/255`
	`indices = np.arange(X.shape[0])`
	`np.random.shuffle(indices)#shuffle all images`
	`X = X[indices]`
	`Y = Y[indices]`
	`Y = to_categorical(Y)`
	`#split dataset into train and test`
	`X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.2)`
	`print("Dataset Image Processing & Normalization Completed")`
	`print("80% images used to train CNN algorithm : "+str(X_train.shape[0]))`
	`print("20% image used to train CNN algorithm : "+str(X_test.shape[0]))`
	
	`Dataset Image Processing & Normalization Completed`
	`80% images used to train CNN algorithm : 1248`
	`20% image used to train CNN algorithm : 312`

Data Augmentation	Applying random transformations to increase dataset diversity.
	`]: #finding & plotting graph of non-addicted and addicted instances`
	`#visualizing class labels count found in dataset`
	`label, count = np.unique(Y, return_counts = True)`
	`print("Benign : "+str(count[0]))`
	`print("Malignant : "+str(count[1]))`
	`print("Normal : "+str(count[2]))`
	`height = count`
	`bars = labels`
	`y_pos = np.arange(len(bars))`
	`plt.figure(figsize = (4, 3))`
	`plt.bar(y_pos, height)`
	`plt.xticks(y_pos, bars)`
	`plt.xlabel("Dataset Class Label Graph")`
	`plt.ylabel("Count")`
	`plt.xticks()`
	`plt.show()`
	
	`Benign : 874`
	`Malignant : 420`
	`Normal : 266`

<table>
<tr>
<td>Batch Normalization</td>
<td>
<pre>
]: #preprocess images like shuffling and normalization
X = X.astype('float32')
X = X/255
indices = np.arange(X.shape[0])
np.random.shuffle(indices)#shuffle all images
X = X[indices]
Y = Y[indices]
Y = to_categorical(Y)
#split dataset into train and test
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.2)
print("Dataset Image Processing & Normalization Completed")
print("80% images used to train CNN algorithm : "+str(X_train.shape[0]))
print("20% image used to train CNN algorithm : "+str(X_test.shape[0]))

Dataset Image Processing & Normalization Completed
80% images used to train CNN algorithm : 1248
20% image used to train CNN algorithm : 312
</pre>
</td>
</tr>
<tr>
<td>CNN Algorithm</td>
<td>
<pre>
In [93]: #ccreating CNN object
cnn_model = Sequential()
#adding CNN2d layer with 32 neurons of size 3 X 3 to filter images 32 times
cnn_model.add(Convolution2D(32, (3 , 3), input_shape = (X_train.shape[1], X_train.shape[2], X_train.shape[3]), activation = 'relu'))
#max pool layer to collect filtered relevant features from previous CNN layer
cnn_model.add(MaxPooling2D(pool_size = (2, 2)))
#adding another layer with relu activation function
#ReLU helps the first hidden layer receive errors from the last layers to adjust all weights between layers
cnn_model.add(Convolution2D(32, (3, 3), activation = 'relu'))
cnn_model.add(MaxPooling2D(pool_size = (2, 2)))
cnn_model.add(Flatten())
#adding LSTM as RNN layer
cnn_model.add(RepeatVector(2))
cnn_model.add(LSTM(32, activation = 'relu'))#=================adding RNN LSTM
#defining output layer with extra softmax layer which will divide each class prediction into probabilities and the
#class with highest probability will be best prediction and help in enhancing accuracy
cnn_model.add(Dense(units = 256, activation = 'relu'))
cnn_model.add(Dense(units = y_train.shape[1], activation = 'softmax'))
#compile the model
cnn_model.compile(optimizer = 'adam', loss = 'categorical_crossentropy', metrics = ['accuracy'])
#train and load the model
if os.path.exists("model/cnn_weights.hdf5") == False:
    model_check_point = ModelCheckpoint(filepath='model/cnn_weights.hdf5', verbose = 1, save_best_only = True)
    hist = cnn_model.fit(X_train, y_train, batch_size = 32, epochs = 15, validation_data=(X_test, y_test), callbacks=[model_check_point], verbose=1)
    f = open('model/cnn_history.pckl', 'wb')
    pickle.dump(hist.history, f)
    f.close()
else:
    cnn_model.load_weights("model/cnn_weights.hdf5")
#perform prediction on test data using cnn model
predict = cnn_model.predict(X_test)
predict = np.argmax(predict, axis=1)
y_test1 = np.argmax(y_test, axis=1)
#call this function to true test labels and predicted labels to calculate accuracy and other metrics
calculateMetrics("CNN with Softmax", y_test1, predict)

CNN with Softmax Accuracy : 99.35897435897436
CNN with Softmax Precision : 99.43602693602695
CNN with Softmax Recall   : 99.43602693602695
CNN with Softmax FSCORE   : 99.43602693602695
</pre>
</td>
</tr>
</table>