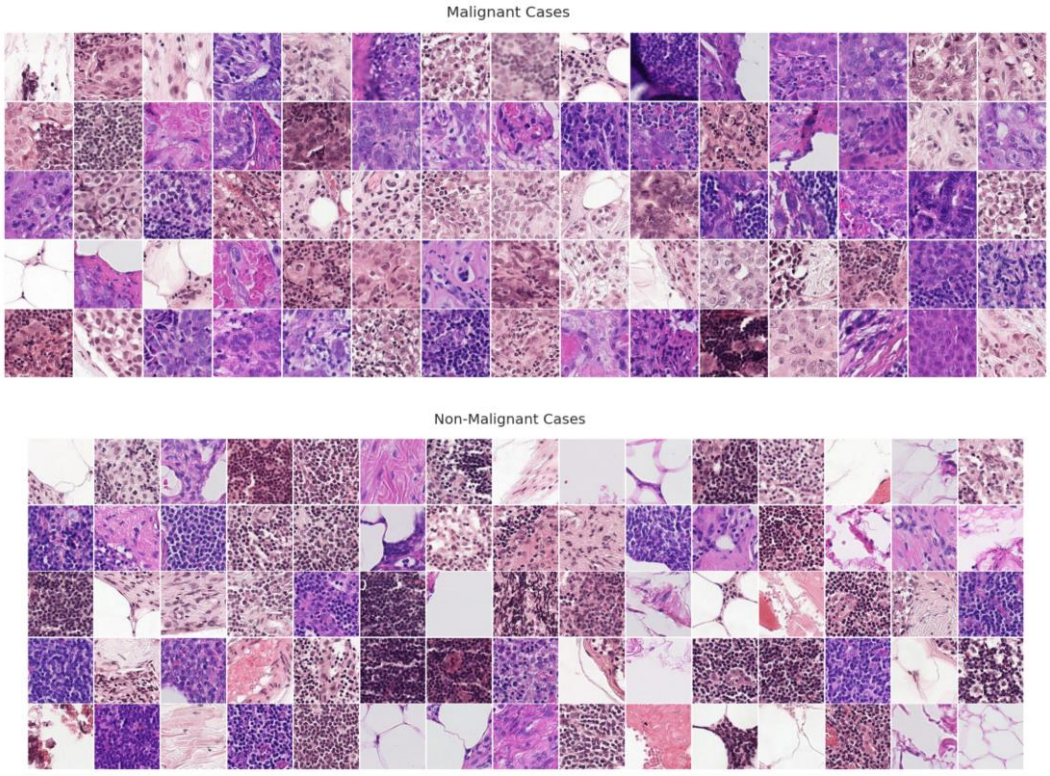
Pattern Recognition and Machine Learning

Histopathologic Cancer Detection
(Major Project)

By Jatin Lohar (B21CS091)
Himanshu Gaurav (B21EE025)
Himesh Dhaka (B21EE026)

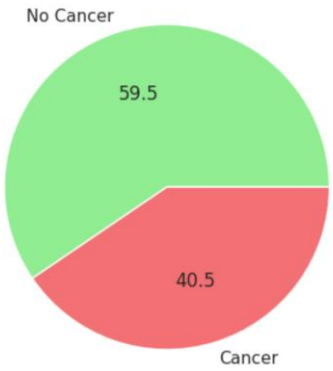
1. Introduction

To improve cancer detection, early diagnosis is essential, which can only be achieved through an effective detection system. Histopathology generates microscopic images of tissue biopsies, which can be used to develop computer-assisted detection systems. Manual detection is a laborious and error-prone task, as cellular structures are often complex and viewed from different angles. The objective is to differentiate between benign and malignant tumours, as the latter require prompt treatment to avoid further complications. This is a binary classification problem that can be addressed using various machine learning techniques.



Dataset: The "histopathologic-cancer-detection.zip" file consists of two folders, "train" and "test," containing a total of 275,000 microscopic images of tissue samples. Each image has a unique identifier. The "train" folder contains 220,025 images, and the "test" folder contains over 55,000 images. Additionally, the "train_labels.csv" file contains the correct classifications for the images in the "train" folder. Furthermore, a subset of the "train"

folder was selected randomly to create the "Random Sample" dataset, which contains 5,500 images. Among the images in the "train" dataset, 59.5% of them are positive for cancer, while the remaining 40.5% are negative for cancer.



2. Pre-processing

We implement a Py Torch dataset class called pytorch_data that takes in a data directory path, a transform object, and a data type (either "train" or "test"). The class randomly selects 10,000 image files from the specified data directory, loads their corresponding labels from a CSV file, and stores both the file paths and their labels as class variables. The __len__ method returns the total number of images in the dataset, and the __getitem__ method returns the transformed image and its corresponding label given an index. The transform applied to the image resizes it to a 48x48 size and converts it to a tensor.

At last, we instantiate the dataset object and retrieve the image and label at index 10 for demonstration purposes. It prints the shape of the image tensor and the minimum and maximum pixel values of the tensor.

Image transformation and augmentation are crucial for building deep learning models along with pretrained models. These techniques can help expand and enhance the dataset for better model performance. Common image transformations include flipping, rotation, resizing, and normalization. These transformations can be applied without altering the labels, which is useful for binary classification models. The torchvision module provides a set of image transformations that can be used during the training process. Some of the common training data augmentations include random horizontal and vertical flipping, random rotation, random resizing and cropping, and conversion to tensor format with normalization.

3. Training the Model

The provided code is for training a CNN model on a given dataset and then using the trained model for inference on a separate test dataset. The training parameters are set in a dictionary called params_train, which includes the data loaders, optimizer, loss function, weight path, and number of epochs. The training process is run with the train_val() function, and the loss and accuracy history are saved in loss_hist and metric_hist, respectively. These histories are then plotted using plotly.

A separate class called pytorchdata_test is defined for the test dataset, which is similar to the pytorch_data class used for the training and validation datasets. After the model is trained, it is used to make predictions on the test dataset using the inference function, and the accuracy is calculated by comparing the predicted labels to the ground truth labels.

We'll be training the classifier for **200 iterations** using the **Adam optimiser** & the **negative log likelihood loss** function

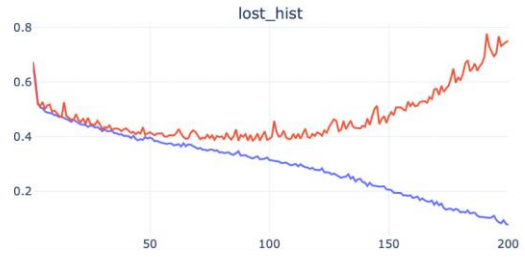
4. Results and Analysis

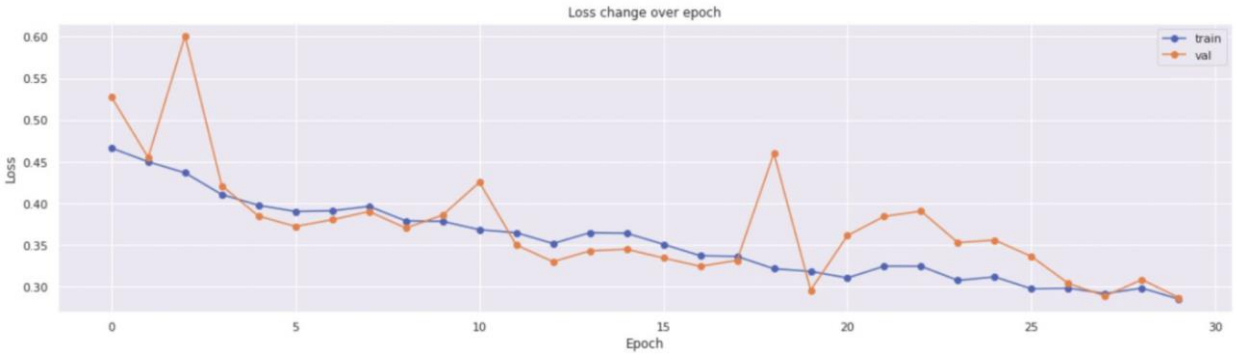
The best validation accuracy of **83.15 %** was achieved by training a CNN on the sampled dataset, which outperformed all previous models we attempted. This confirms our decision to implement a CNN. We then trained the same architecture on the complete dataset for 30 epochs, which resulted in a testing accuracy of **79.00 %**. Unfortunately, due to limited resources and long training times, we were unable to perform hyperparameter tuning and model refinement.

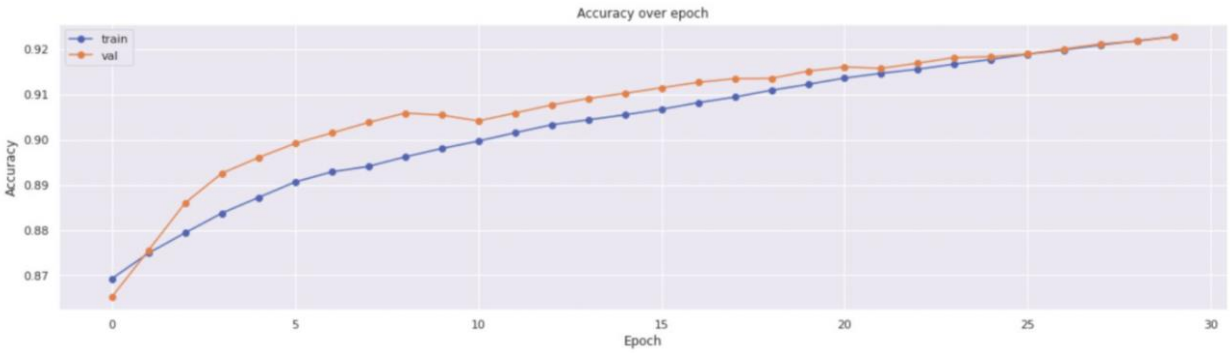
A CNN undergoes two major transformations during its training. The first transformation is convolution, in which the pixels are convolved with a filter or kernel. The second transformation is subsampling, which can take various forms, such as max pooling, min pooling, and average pooling, and is applied as necessary. The pooling layer reduces the dimensionality of the data, which is useful for reducing overfitting. The output is then passed to a fully connected layer for efficient classification after using a combination of convolution and pooling layers.

Model	Precision	Recall	F1 score
Without PCA and LDA			
Linear SVC	0.59	0.59	0.59
KNN	0.64	0.63	0.59
RBF SVC	0.80	0.78	0.78
Light GBM	0.78	0.76	0.77
With PCA (0.9) and LDA			
Linear SVC	0.82	0.81	0.82
KNN	0.81	0.81	0.81
RBF SVC	0.82	0.82	0.82
Light GBM	0.82	0.83	0.83

Dimensionality decrease improves model efficiency, as seen above. The dip in f1-scores is minor, and the reduced mathematical complexity may compensate.







5. Contributions

Individual contributions can be mapped as follows:

Jatin Lohar: CNN, PCA & LDA, Linear SVC, KNN, Light GBM, RBF SVC

Himanshu Gaurav: Report making, PCA &LDA, KNN

Himesh Dhaka: CNN, Pre-processing and Plots