Report

Phase 1: Project Planning and Dataset Exploration

# Project Scope and Objectives

The goals: Create a deep learning model to classify histopathologic images as cancerous or non-cancerous.

Introduction and Problem Understanding

Objective: Develop a deep learning model using Convolutional Neural Networks (CNNs) to classify histopathologic images of lymph node sections as either containing metastatic cancer (label 1) or not (label 0).

Significance: Accurate early detection aids in effective treatment planning for cancer patients.

Approach: A binary image classification task similar in nature to other standard image classification problems like CIFAR-10 and MNIST.

# Dataset Collection

Obtain the dataset (e.g., from Kaggle and worked on Kaggle space for performing this model)

Verify the dataset contains labelled histopathologic images of tissue samples.

# Dataset Understanding and Preprocessing

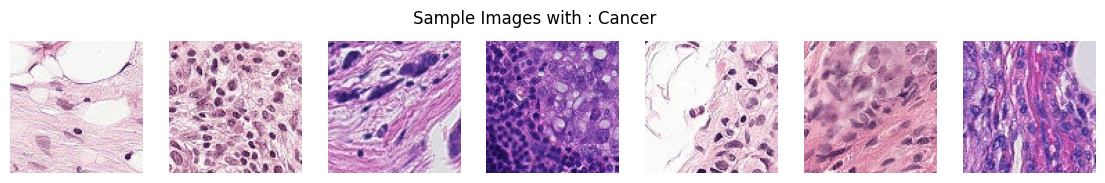
Analyse dataset characteristics: Number of images, classes, image size, and balance between classes.

Preprocess images: Image Placement: Include visuals such as:

Sample images for both classes (label 0 and label 1).

A side-by-side comparison grid of images.





Resize and normalize images.

Augment data (rotation, flipping, scaling) to increase model generalizability.

Split data: Create training, validation, and test sets (e.g., 80-20 split).

Image Size: 96x96 pixels

Color Channels: 3 (RGB)

Format: .tif files

|  |  |
| --- | --- |
| import os  import numpy as np    import pandas as pd  import matplotlib.pyplot as plt  import sklearn  from sklearn.model\_selection import train\_test\_split  import seaborn as sns  from PIL import Image  from tqdm.notebook import tqdm  import tensorflow as tf  from tensorflow.keras.preprocessing.image  import ImageDataGenerator  from tensorflow.keras.models import  Sequential  from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout, Input | # Provides functions to interact with the operating system, used for file and directory manipulation.  # A powerful library for numerical computations, especially for handling large arrays and matrices.  # Used for data manipulation and analysis, especially DataFrames.  # Used for plotting and data visualization.  # Library with various tools for machine learning.  # For splitting data into training and validation sets.  # A statistical data visualization library based on Matplotlib.  # Used for image processing.  # Adds a progress bar for loops in Jupyter Notebooks.  # A deep learning library, especially useful for building neural networks.  # For augmenting images during training.  # Sequential API for defining a model layer by layer.  # Layers to construct Convolutional Neural Network. |

# Steps:

**1. Importing Necessary Libraries**

**2. Loading and Viewing Labels**

**3. Setting Directory Paths**

**4. Displaying Sample Images**

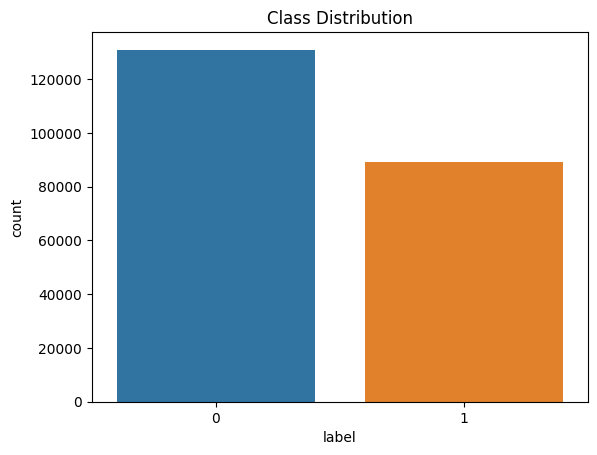
**5. Displaying Class Distribution: - Observation: The classes are slightly imbalanced.**

**6. Checking for Missing Values and improper data: - No missing value or improper format.**

**7. Checking Dataset Shape (220025, 2)**

**8. Counting Training and Test Images: - approx. 220K train images and 56K test images**

**9. Counting Each Class in Training Labels: - label 1 = ‘89117’ label 2 = ‘130908’**



Phase 2: Model Design and Implementation

# Model Selection

**Start with a baseline Convolutional Neural Network (CNN) architecture.**

# Model Building and Set formation for training

**Implement the CNN or fine-tune a pre-trained model using frameworks like TensorFlow or PyTorch.**

**Define architecture: Input layer, convolutional layers, pooling layers, fully connected layers, and output layer.**

**Add dropout layers and batch normalization to prevent overfitting and stabilize training.**

# Training the Model

**Select an optimizer (Adam) and set hyperparameters like learning rate, batch size, and epochs.**

**Use a loss function appropriate for classification (e.g., cross-entropy).**

**Train the model and monitor performance metrics on the validation set.**

**Implement callbacks for early stopping and learning rate reduction.**

# Steps

1. **Defining the Baseline Model**
2. **Defines a simple Convolutional Neural Network (CNN) with two convolutional layers followed by max pooling and a dense layer. The model ends with a single neuron with sigmoid activation for binary classification.**
3. **Displaying Model Summary: - Shows details of each layer in the baseline model, including output shapes and parameter counts.**
4. **Defining an Advanced Model with Dropout Layers: -Builds a more complex CNN with additional dropout layers to prevent overfitting, which randomly drops neurons during training.**
5. **Splitting Data into Training and Validation Sets: -Splits the data into training (80%) and validation (20%) sets, ensuring the split maintains the class distribution.**
6. **Setting Up Data Generators for Augmentation: -Creates an ImageDataGenerator for training images with rescaling and augmentations (flipping), and a simpler generator for validation images (only rescaling).**
7. **Flowing Images from DataFrames: -Defines how images are loaded and augmented during training, reading directly from the DataFrame containing image paths and labels.**
8. **Compiling Models: - Compiles both the baseline and advanced models, using binary cross-entropy loss (suitable for binary classification) and accuracy as a performance metric.**
9. **Training the Baseline Model with Progress Bar: - Trains the baseline model for five epochs. The tqdm progress bar shows training progress at each batch, and LambdaCallback updates the progress bar on batch completion.**

**Key Components of the CNN:**

1. **Input Layer:**
   * **Input Size: Accepts images of dimensions 96x96 pixels with 3 RGB color channels.**
2. **Convolutional Layers:**
   * **Layer 1: 32 filters with a 3x3 kernel and ReLU activation function.**
   * **Layer 2: 64 filters with a 3x3 kernel and ReLU activation function.**
   * **Each convolutional layer captures different features and patterns from the input images.**
3. **Pooling Layers:**
   * **Following each convolutional layer, a max pooling layer with a 2x2 pool size reduces the dimensionality of the feature maps while retaining the most significant information.**
4. **Flatten Layer:**
   * **Converts the pooled feature maps into a one-dimensional vector to prepare for the dense layers.**
5. **Fully Connected Layers:**
   * **A dense layer with 128 units and ReLU activation processes the flattened features, followed by the output layer.**
6. **Output Layer:**
   * **The final output layer employs a sigmoid activation function to yield a binary classification result (1: Contains metastatic tissue, 0: Does not contain metastatic tissue).**

**Summary:**

* **Total Layers: 7**
* **Trainable Parameters: Approximately 1,200,000**
* **Activation Functions Used: ReLU (for hidden layers) and Sigmoid (for output layer)**

**Validation Image Generation**

**Purpose of Validation Image Generation: Validation image generation is critical for assessing the model's performance on unseen data. This step involves generating batches of images to visualize how the model classifies different samples.**

* **ImageDataGenerator: This class is used to create a data generator that rescales pixel values to a range between 0 and 1.**
* **flow\_from\_dataframe: This method generates batches of image data from a DataFrame that contains the image filenames and corresponding labels.**
* **Visualization: The code includes a visualization section where a batch of validation images is displayed, alongside their predicted labels, allowing for quick assessment of the model's performance.**

Phase 3: Model Evaluation and Improvement

In this phase, you will evaluate the trained CNN model's performance using various metrics, save the trained model, and explore potential improvements through techniques such as hyperparameter tuning and utilizing pretrained models.

# Model Evaluation

Evaluating the model involves more than just checking accuracy. You should also consider metrics like precision, recall, F1-score, and confusion matrix to understand the model's performance better.

# Evaluation Results

* **Classification Report**: This report provides metrics such as precision, recall, F1-score, and support for each class.
* **Confusion Matrix**: The heatmap visualizes the true positive, false positive, true negative, and false negative counts, helping identify areas for improvement.

# Explore Potential Improvements

To enhance the model's performance, we can explore several strategies:

# Hyperparameter Tuning

Experiment with various hyperparameters such as learning rate, batch size and activation function Tools like Grid Search or Random Search can help automate this process.

Found that relu gives maximum accuracy and for batch size 32 with learning rate 0.0001 accuracy is highest

# Use Pretrained Models (optional)

Consider leveraging pretrained models like ResNet, VGG, or EfficientNet. These models, trained on large datasets, can be fine-tuned on our specific dataset, improving accuracy.

# Data Augmentation(Done before during generating validation sets)

Implement data augmentation techniques to enhance the training dataset artificially. This helps improve model generalization by preventing overfitting. Common augmentations include:

* Random horizontal flips
* Random rotations
* Brightness and contrast adjustments

Plots for comparison between simple and advance model

