Final Project Report

Team ID : SWTID1720011518

1. Introduction

a. Project Overview

Develop a deep learning model for curated colon disease classification from medical imagingdata. By analyzing colonoscopy images and patient records,this project aims toaccurately classify various colon diseases, aiding in early detection, treatment planning, and improving patient outcomes.

b. Objectives

i. know fundamental concepts and techniques of Convolutional NeuralNetwork.

ii. gain a broad understanding of image data.

iii. Know how to pre-process/clean the data usingdifferent data preprocessingtechniques.

iv. know how to builda web application using Flask framework.

2. Project Initialization and planningphase

Define ProblemStatements (Customer ProblemStatement Template):

Create a deep learning model to classify colon diseases using medical imaging data in a curated manner. This initiative intendsto reliably categorize diverse colon disordersthrough the analysisofpatient data and colonoscopy images, therefore facilitating early identification, treatment planning,and improved patient outcomes. Use the deep learning model to aid in the diagnosis of colon disorders by medical practitioners. Byproviding rapid and precise illness categorization, we may enhance patient care, expedite treatment decisions, and increase diagnostic accuracy.

Reference:
https://miro.com/app/board/uXjVKzylESg=/?share_link_id=115441034671

Problem Statement(PS)	**I am (Customer)**	**I'm trying to**	**But**	**Because**	**Which makes me feel**
PS-1	A Doctor/Healthcare Professional	Improve my accuracy of prediction of Colon Disease and improve patient care by providing timely and accurate disease classification.	It is taking me too much time and is causing delay in diagnosing of the disease	I have to go through all the endoscopy reports manually	I'm Very Slow, andalways behind the time

Project Proposal (Proposed Solution) template

This project proposal outlines a solution to address a specific problem. With a clear objective, defined scope, and a concise problemstatement, the proposedsolution details the approach, keyfeatures, and resource requirements, including hardware, software, and personnel.

Project Overview	
Objective	To assist healthcare professionals in diagnosing colon diseases.Enhance diagnostic accuracy, streamline treatment decisions, andimprove patient care by providing timely and accurate disease classification.
Scope	The scope involves collecting and preprocessing high-quality colonoscopy images, developing a robust deep learning model, integrating it intohealthcare systems, andensuring compliance withethical and regulatory standards.
Problem Statement	
Description	Healthcare professionals struggle with accurately diagnosing colon diseases due to inconsistent imaging data and complex medical records. Developing a deep learning model to analyze these data sources can enhance diagnostic precision, support earlydetection, andimprove treatment planning and patient outcomes.
Impact	The solution will significantly improve diagnostic accuracy, enablingearly detection of colon diseases and facilitating timelyinterventions.It will streamline treatment planning, providing healthcare professionals with reliable data for informed decisions. This will enhance patient outcomes and operational

Proposed Solution	
Approach	We will utilize a Kaggle dataset and employ transfer learning for thisproject. By extracting features using various pre-trained models, we will evaluate their performance and select the model that yields the best results. Thisapproach ensures optimal accuracy and efficiency inclassifying colon diseases from medical imaging

Key Features	The key features of the proposed solution include using a comprehensive Kaggle dataset, employing transfer learning for efficient feature extraction, evaluating multiple pre-trained models foroptimal performance, and integrating the best-performing model into healthcare systems. This approach enhances diagnostic accuracy, supports early detection, and streamlines treatment planning for colondiseases.

Resource Requirements

Resource Type	Description	Specification/Allocation
Hardware		
Computing Resources	CPU/GPU specifications,number	Any Basic GPU
Memory	RAM specifications	8 GB
Storage	Disk space for data,models,and logs	512 GB SSD
Software		
Frameworks	Python frameworks	Flask
Libraries	Additional libraries	Tensorflow, Numpy
Development Environment	IDE, version control	Google Colab,Spyder
Data		
Data	Source, size,format	Kaggle dataset (WCE Curated Colon Disease Dataset Deep Learning), 6,000 images

3. Data Collection and Preprocessing Phase

Data Collection Plan & Raw Data SourcesIdentification Template

The model requires labelled data consisting of Colonoscopy Images and associated labels for each image indicating the presence or absence of colon disease and the specific disease type. Such data can be collected from medical institutions, public databases and research collaborators.

Data Collection Plan Template

Section	Description
Project Overview	The project aims to classify the WCE Curated colon diseases based on Wireless Capsule Endoscopy (WCE)images. The project aims to reducetime in detection and classification of the disease.
Data Collection Plan	Data willbe collected froman already existing dataset on Kaggle.
Raw Data Sources Identified	Data can be collected from medical institutions, public databases andresearch collaborators.

Raw Data Sources Template

Source Name	Description	Location/URL	Format	Size	Access Permissions
	The dataset is taken from J. Silva, A. Histace, O. Romain, X. Dray and B. Granado, "Toward				

WCE Curated Colon Disease Dataset Deep Learning	embedded detection of polyps in WCE images for early diagnosis of colorectal cancer", International Journal of Computer Assisted Radiology and Surgery, vol.9, no. 2, pp. 283-293, 2013. DOI:10.1007/s11548-013-0926-3.	https://www.kaggle.com/datasets/francismon/curated-colon-dataset-for-deep-learning	Images	2 GB	Public

Data Collection Plan & Raw Data Sources Identification Template

Elevate your data strategy with the Data Collection plan and the Raw Data Sources report, ensuring meticulous data curation and integrity for informed decision-making in every analysisand decision-making endeavor.

Data Collection Plan Template

Section	Description

Project Overview	Healthcare professionals struggle with accurately diagnosing colon diseases due to inconsistent imaging data and complex medical records. Developing a deep learning model to analyze these data sources can enhance diagnostic precision, support early detection, and improve treatment planning and patient outcomes.
Data Collection Plan	The dataset which is used is a public dataset. It is available on Kagglewith the name"Curated Colon Datasetfor Deep Learning"
Raw Data Sources Identified	The dataset contains medical imaging data in a curated manner. This initiative intends to accurately categorize diverse on these orders through the analysis of patient data and colonoscopy images therefore facilitating earlyidentification treatment planning and improved patientoutcomes

Raw Data Sources Template

Source Name	Description	Location/URL	Format	Size	Access Permissions
Kaggle	WCE Curated Colon Disease Dataset Deep Learning	https://www.kaggle.com/datasets/francismon/curated- colon-dataset-for- deep-learning	Images	2 GB	Public

Data Quality Report Template

The Data QualityReport Template will summarize data quality issues from the selected source,including severity levels and resolution plans. It will aid in systematically identifying and rectifying data discrepancies.

Data Source	Data QualityIssue	Severity	Resolution Plan
WCE Curated Colon diseases.	Sizes and orientation of images were different. Contrast in colorof images wasdifferent.	Low	Rescaling, Normalization and Gray scaling.
WCE Curated Colon diseases.	Borders of disease patcheswerehard to too sometimes.	Moderate	Denoising with Gaussian blur and Edge detection with Canny edgedetector.

Preprocessing

The images will be preprocessed by resizing, normalizing, augmenting, denoising, adjusting contrast, detectingedges, converting color space, cropping, batch normalizing, and whiteningdata. These steps will enhance data quality, promote model generalization, and improve convergence during neural networktraining, ensuring robust and efficientperformance acrossvariouscomputer vision tasks.

Section	Description
Data Overview	The images have been captured via Wireless Capsule Endoscopy (WCE).There are 3200training images, 800testingimages and 2000 validation images. All are divided into fourcategories: Normal, Ulcerative Colitis, Polyps, Esophagitis.
Resizing	Resize imagesto a specified target size.

Normalization	Normalize pixelvalues to a specific range.
Data Augmentation	Apply augmentation techniques such as flipping, rotation,shifting, zooming, or shearing.
Denoising	Apply denoising filters to reducenoise in the images.
Edge Detection	Apply edgedetection algorithms to highlight prominent edgesin theimages.

Color SpaceConversion	Convert imagesfrom one colorspace to another.
Image Cropping	Crop imagesto focus on the regions containing objects of interest.
Batch Normalization	Apply batch normalization to the inputof each layerin the neural network.
Data Preprocessing Code Screenshots	

Loading Data	`# Define paths`
	`train_path = "/content/train"`
	`test_path = "/content/test"`
	`valid_path = "/content/val"`

Resizing	`# For testing and validation, only rescaling is applied`
	`test_datagen = ImageDataGenerator(rescale=1./255)`
	`valid_datagen = ImageDataGenerator(rescale=1./255)`

Normalization	`train_datagen = ImageDataGenerator(`
	`rescale=1./255,`
	`zoom_range=0.2,`
	`shear_range=0.2,`
	`preprocessing_function=preprocess_input  # VGG16 preprocessing (mean subtraction)`
	`)`

Data Augmentation	`train_datagen = ImageDataGenerator(`
	`rescale=1./255,`
	`zoom_range=0.2,`
	`shear_range=0.2,`
	`preprocessing_function=preprocess_input  # VGG16 preprocessing (mean subtraction)`
	`)`

Denoising	`def preprocess_image(img):`
	`    # Denoising with Gaussian blur`
	`    img = cv2.GaussianBlur(img, (5, 5), 0)`

Edge Detection	`# Edge detection with Canny edge detector` `edges = cv2.Canny(gray, 100, 200)`
Color SpaceConversion	`# Convert to grayscale` `gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)`

Image Cropping	`# Image cropping (adjust cropping dimensions as needed)` `crop_img = img[50:150, 50:150]`

Batch Normalization	`# Flow data from directories`
	`train_generator = train_datagen.flow_from_directory(`
	`    train_path,`
	`    target_size=(224, 224),`
	`    batch_size=20,`
	`    class_mode='categorical'`

4. Model Development Phase

Model Selection Report

In the model selection report for future deep learning and computer vision projects, various architectures, such as CNNs or RNNs, will be evaluated. Factors such as performance, complexity, and computational requirements will be considered to determine the most suitablemodel for the task at hand.

Model Selection Report:

Model	Description

VGG16	By using feature extraction, we haveused the VGG16pre-trained model. One Dense and one Flatten layer is used. The loss function is categorical_crossentropy, optimizer is “adam” and metrics forevaluation is accuracy. For 5 epochs: The Training Loss is: 0.0272 The Training Accuracy is: 0.9909 The Validation Loss is: 0.0190The Validation Accuracy is: 0.9907

Resnet50	By using feature extraction, we haveused the Resnet50 pre-trained model. One Dense and one Flatten layer is used. The loss function is categorical_crossentropy, optimizer is “adam” and metrics for evaluation is accuracy. For 5 epochs: The Training Loss is: 0.3833 The Training Accuracy is: 0.8625 The Validation Loss is: 0.2326The Validation Accuracy is: 0.9047

InceptionV3	By using feature extraction, we have used the InceptionV3 pre-trained model. One Dense and one Flatten layer is used. The loss function is categorical_crossentropy, optimizer is “adam” and metrics forevaluation is accuracy. For 5 epochs: The Training Loss is: 0.2585 The Training Accuracy is: 0.9850 The Validation Loss is: 4.9814The Validation Accuracy is: 0.8537

Initial Model TrainingCode, Model Validation and Evaluation Report

The initial model training code will be showcased in the future through a screenshot. The model validation and evaluation report will include a summaryand training and validation performancemetrics for multiple models, presented through respective screenshots.

Initial Model Training Code (5 marks):

VGG16:

```
for layer in vgg.layers:
  layer.trainable = False
```

```
x = Flatten()(vgg.output)
```

```
output = Dense(4,activation = "softmax")(x)
```

```
vgg16 = Model(vgg.input , output)
```

```
vgg16.summary()
```

```
Model: "model"

 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 224, 224, 3)]     0

 block1_conv1 (Conv2D)       (None, 224, 224, 64)      1792

 block1_conv2 (Conv2D)       (None, 224, 224, 64)      36928
```

```
vgg16.compile(optimizer = "adam",loss = "categorical_crossentropy",metrics = ["accuracy"])
```

```
history = vgg16.fit(train , validation_data= test,epochs = 5)
```

```
Epoch 1/5
160/160 [==============================] - 107s 622ms/step - loss: 0.2567 - accuracy: 0.9134 - val_loss: 0.1309 - val_accuracy: 0.9469
Epoch 2/5
160/160 [==============================] - 100s 624ms/step - loss: 0.0870 - accuracy: 0.9688 - val_loss: 0.0259 - val_accuracy: 0.9912
Epoch 3/5
160/160 [==============================] - 100s 628ms/step - loss: 0.0551 - accuracy: 0.9806 - val_loss: 0.1200 - val_accuracy: 0.9481
Epoch 4/5
160/160 [==============================] - 82s 514ms/step - loss: 0.0398 - accuracy: 0.9847 - val_loss: 0.1132 - val_accuracy: 0.9522
Epoch 5/5
160/160 [==============================] - 81s 509ms/step - loss: 0.0272 - accuracy: 0.9909 - val_loss: 0.0190 - val_accuracy: 0.9937
```

Resnet50:

```
for layer in resnet50.layers:
  layer.trainable = False
```

```
x1 = Flatten()(resnet50.output)
```

```
output = Dense(4,activation = "softmax")(x1)
```

```
resnet = Model(resnet50.input , output)
```

```
resnet.summary()
```

```
 conv5_block2_2_bn (BatchNo  (None, 7, 7, 512)            2048      ['conv5_block2_2_conv[0][0]']
 rmalization)

 conv5_block2_2_relu (Activ  (None, 7, 7, 512)            0         ['conv5_block2_2_bn[0][0]']
 ation)

 conv5_block2_3_conv (Conv2  (None, 7, 7, 2048)           1050624   ['conv5_block2_2_relu[0][0]']
 D)

 conv5_block2_3_bn (BatchNo  (None, 7, 7, 2048)           8192      ['conv5_block2_3_conv[0][0]']
```

```
resnet.compile(loss = "categorical_crossentropy", optimizer = "adam" , metrics = ["accuracy"])
```

```
history3 = resnet.fit(train , validation_data = test , epochs = 5)
```

```
Epoch 1/5
160/160 [==============================] - 81s 509ms/step - loss: 0.4264 - accuracy: 0.8394 - val_loss: 0.3856 - val_accuracy: 0.8422
Epoch 2/5
160/160 [==============================] - 81s 504ms/step - loss: 0.5056 - accuracy: 0.8156 - val_loss: 0.3810 - val_accuracy: 0.8481
Epoch 3/5
160/160 [==============================] - 80s 500ms/step - loss: 0.5100 - accuracy: 0.8231 - val_loss: 0.3660 - val_accuracy: 0.8537
Epoch 4/5
160/160 [==============================] - 81s 506ms/step - loss: 0.3765 - accuracy: 0.8594 - val_loss: 0.2215 - val_accuracy: 0.9191
Epoch 5/5
160/160 [==============================] - 80s 500ms/step - loss: 0.3833 - accuracy: 0.8625 - val_loss: 0.2326 - val_accuracy: 0.9047
```

InceptionV3:

```
for layer in inception_v3.layers:
  layer.trainable = False
```

```
x3 = Flatten()(inception_v3.output)
```

```
output3 = Dense(4,activation = "softmax")(x3)
```

```
inception = Model(inception_v3.input, output3)
```

```
inception.summary()
```

```
conv2d_88 (Conv2D)          (None, 8, 8, 384)     442368    ['activation_86[0][0]']

conv2d_91 (Conv2D)          (None, 8, 8, 384)     442368    ['activation_90[0][0]']

conv2d_92 (Conv2D)          (None, 8, 8, 384)     442368    ['activation_90[0][0]']

average_pooling2d_8 (Avera  (None, 8, 8, 2048)    0         ['mixed9[0][0]']
gePooling2D)

conv2d_85 (Conv2D)          (None, 8, 8, 320)     655360    ['mixed9[0][0]']

batch_normalization_87 (Ba  (None, 8, 8, 384)     1152      ['conv2d_87[0][0]']
```

```
inception.compile(loss = "categorical_crossentropy", optimizer = "adam", metrics = ["accuracy"])
```

```
history4 = inception.fit(train, validation_data = test,epochs = 5)
```

```
Epoch 1/5
160/160 [==============================] - 106s 614ms/step - loss: 2.6529 - accuracy: 0.8931 - val_loss: 0.8691 - val_accuracy: 0.9375
Epoch 2/5
160/160 [==============================] - 92s 577ms/step - loss: 0.5101 - accuracy: 0.9691 - val_loss: 0.6682 - val_accuracy: 0.9525
Epoch 3/5
160/160 [==============================] - 91s 564ms/step - loss: 0.3904 - accuracy: 0.9747 - val_loss: 1.3684 - val_accuracy: 0.9362
Epoch 4/5
160/160 [==============================] - 91s 566ms/step - loss: 0.4131 - accuracy: 0.9778 - val_loss: 0.6467 - val_accuracy: 0.9675
Epoch 5/5
160/160 [==============================] - 90s 561ms/step - loss: 0.2585 - accuracy: 0.9850 - val_loss: 4.9814 - val_accuracy: 0.8537
```

Model Validation and Evaluation Report(5 marks):

Model	Summa	Training andValidationPerformance Metrics

VGG16

```
vgg.summary()

Model: "vgg16"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 224, 224, 3)]     0

 block1_conv1 (Conv2D)       (None, 224, 224, 64)      1792

 block1_conv2 (Conv2D)       (None, 224, 224, 64)      36928

 block1_pool (MaxPooling2D)  (None, 112, 112, 64)      0

 block2_conv1 (Conv2D)       (None, 112, 112, 128)     73856

 block2_conv2 (Conv2D)       (None, 112, 112, 128)     147584

 block2_pool (MaxPooling2D)  (None, 56, 56, 128)       0

 block3_conv1 (Conv2D)       (None, 56, 56, 256)       295168

 block3_conv2 (Conv2D)       (None, 56, 56, 256)       590080

 block3_conv3 (Conv2D)       (None, 56, 56, 256)       590080
```

```
 block4_pool (MaxPooling2D)  (None, 14, 14, 512)       0

 block5_conv1 (Conv2D)       (None, 14, 14, 512)       2359808

 block5_conv2 (Conv2D)       (None, 14, 14, 512)       2359808

 block5_conv3 (Conv2D)       (None, 14, 14, 512)       2359808

 block5_pool (MaxPooling2D)  (None, 7, 7, 512)         0

 flatten (Flatten)           (None, 25088)             0

 dense (Dense)               (None, 4)                 100356

=================================================================
Total params: 14815044 (56.51 MB)
Trainable params: 100356 (392.02 KB)
Non-trainable params: 14714688 (56.13 MB)
_________________________________________________________________
```

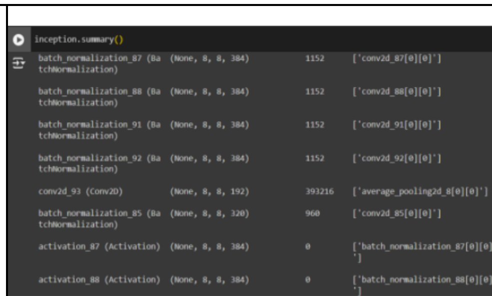
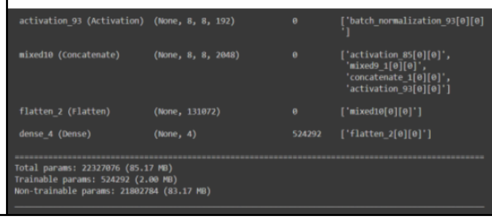
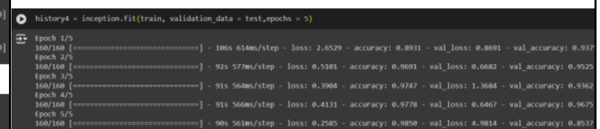
```
history = vgg16.fit(train , validation_data= test,epochs = 5)

Epoch 1/5
160/160 [==============================] - 107s 622ms/step - loss: 0.2567 - accuracy: 0.9134 - val_loss: 0.1309 - val_accuracy: 0.9469
Epoch 2/5
160/160 [==============================] - 100s 624ms/step - loss: 0.0870 - accuracy: 0.9688 - val_loss: 0.0259 - val_accuracy: 0.9912
Epoch 3/5
160/160 [==============================] - 100s 628ms/step - loss: 0.0551 - accuracy: 0.9806 - val_loss: 0.1200 - val_accuracy: 0.9481
Epoch 4/5
160/160 [==============================] - 82s 514ms/step - loss: 0.0398 - accuracy: 0.9847 - val_loss: 0.1132 - val_accuracy: 0.9522
Epoch 5/5
160/160 [==============================] - 81s 509ms/step - loss: 0.0272 - accuracy: 0.9909 - val_loss: 0.0190 - val_accuracy: 0.9937
```

Resnet 50

```
[ ] resnet.summary()

 conv5_block2_2_bn (BatchNo  (None, 7, 7, 512)            2048      ['conv5_block2_2_conv[0][0]']
 rmalization)

 conv5_block2_2_relu (Activ  (None, 7, 7, 512)            0         ['conv5_block2_2_bn[0][0]']
 ation)

 conv5_block2_3_conv (Conv2  (None, 7, 7, 2048)           1050624   ['conv5_block2_2_relu[0][0]']
 D)

 conv5_block2_3_bn (BatchNo  (None, 7, 7, 2048)           8192      ['conv5_block2_3_conv[0][0]']
 rmalization)

 conv5_block2_add (Add)      (None, 7, 7, 2048)           0         ['conv5_block1_out[0][0]',
                                                                     'conv5_block2_3_bn[0][0]']

 conv5_block2_out (Activati  (None, 7, 7, 2048)           0         ['conv5_block2_add[0][0]']
 on)

 conv5_block3_1_conv (Conv2  (None, 7, 7, 512)            1049088   ['conv5_block2_out[0][0]']
 D)

 conv5_block3_1_bn (BatchNo  (None, 7, 7, 512)            2048      ['conv5_block3_1_conv[0][0]']
 rmalization)

 conv5_block3_1_relu (Activ  (None, 7, 7, 512)            0         ['conv5_block3_1_bn[0][0]']
 ation)
```

```
 conv5_block3_3_bn (BatchNo  (None, 7, 7, 2048)           8192      ['conv5_block3_3_conv[0][0]']
 rmalization)

 conv5_block3_add (Add)      (None, 7, 7, 2048)           0         ['conv5_block2_out[0][0]',
                                                                     'conv5_block3_3_bn[0][0]']

 conv5_block3_out (Activati  (None, 7, 7, 2048)           0         ['conv5_block3_add[0][0]']
 on)

 flatten_1 (Flatten)         (None, 100352)               0         ['conv5_block3_out[0][0]']

 dense_1 (Dense)             (None, 4)                    401412    ['flatten_1[0][0]']
```

```
history3 = resnet.fit(train , validation_data = test , epochs = 5)

Epoch 1/5
160/160 [==============================] - 81s 509ms/step - loss: 0.4264 - accuracy: 0.8394 - val_loss: 0.3856 - val_accuracy: 0.84
Epoch 2/5
160/160 [==============================] - 81s 504ms/step - loss: 0.5056 - accuracy: 0.8156 - val_loss: 0.3810 - val_accuracy: 0.84
Epoch 3/5
160/160 [==============================] - 80s 500ms/step - loss: 0.5100 - accuracy: 0.8231 - val_loss: 0.3660 - val_accuracy: 0.85
Epoch 4/5
160/160 [==============================] - 81s 506ms/step - loss: 0.3765 - accuracy: 0.8594 - val_loss: 0.2215 - val_accuracy: 0.91
Epoch 5/5
160/160 [==============================] - 80s 500ms/step - loss: 0.3833 - accuracy: 0.8625 - val_loss: 0.2326 - val_accuracy: 0.90
```

InceptionV3	 	

```
inception.summary()
batch_normalization_87 (Ba  (None, 8, 8, 384)    1152     ['conv2d_87[0][0]']
tchNormalization)

batch_normalization_88 (Ba  (None, 8, 8, 384)    1152     ['conv2d_88[0][0]']
tchNormalization)

batch_normalization_91 (Ba  (None, 8, 8, 384)    1152     ['conv2d_91[0][0]']
tchNormalization)

batch_normalization_92 (Ba  (None, 8, 8, 384)    1152     ['conv2d_92[0][0]']
tchNormalization)

conv2d_93 (Conv2D)          (None, 8, 8, 192)    393216   ['average_pooling2d_8[0][0]']

batch_normalization_85 (Ba  (None, 8, 8, 320)    960      ['conv2d_85[0][0]']
tchNormalization)

activation_87 (Activation)  (None, 8, 8, 384)    0        ['batch_normalization_87[0][0]
                                                          ']

activation_88 (Activation)  (None, 8, 8, 384)    0        ['batch_normalization_88[0][0]
                                                          ']
```

```
activation_93 (Activation)  (None, 8, 8, 192)    0        ['batch_normalization_93[0][0]
                                                          ']

mixed10 (Concatenate)       (None, 8, 8, 2048)   0        ['activation_85[0][0]',
                                                           'mixed9_1[0][0]',
                                                           'concatenate_1[0][0]',
                                                           'activation_93[0][0]']

flatten_2 (Flatten)         (None, 131072)       0        ['mixed10[0][0]']

dense_4 (Dense)             (None, 4)            524292   ['flatten_2[0][0]']

==================================================================================
Total params: 22327076 (85.17 MB)
Trainable params: 524292 (2.00 MB)
Non-trainable params: 21802784 (83.17 MB)
```

```
history4 = inception.fit(train, validation_data = test,epochs = 5)
Epoch 1/5
160/160 [==============================] - 106s 614ms/step - loss: 2.6529 - accuracy: 0.8931 - val_loss: 0.8691 - val_accuracy: 0.937
Epoch 2/5
160/160 [==============================] - 92s 577ms/step - loss: 0.5101 - accuracy: 0.9691 - val_loss: 0.6682 - val_accuracy: 0.9525
Epoch 3/5
160/160 [==============================] - 91s 564ms/step - loss: 0.3904 - accuracy: 0.9747 - val_loss: 1.3684 - val_accuracy: 0.9362
Epoch 4/5
160/160 [==============================] - 91s 566ms/step - loss: 0.4131 - accuracy: 0.9778 - val_loss: 0.6467 - val_accuracy: 0.9675
Epoch 5/5
160/160 [==============================] - 90s 561ms/step - loss: 0.2585 - accuracy: 0.9850 - val_loss: 4.9814 - val_accuracy: 0.8537
```

5. Model Optimization and Tuning Phase

Model Optimization and Tuning Phase

The Model Optimization and Tuning Phase involves refining neural network models for peak performance. It includes optimized model code, fine-tuning hyperparameters, comparing performance metrics, and justifying the final model selection for enhanced predictive accuracyand efficiency.

Hyperparameter Tuning Documentation (8 Marks):

Model	Tuned Hyperparameters

VGG16

```
tuner.search(train, epochs=20, callbacks=[stop_early])

# Get the optimal hyperparameters
best_hps=tuner.get_best_hyperparameters(num_trials=1)[0]

print(f"""
The hyperparameter search is complete. The optimal learning rate for the optimizer
is {best_hps.get('learning_rate')}.
""")
```

```
Trial 3 Complete [00h 02m 00s]
accuracy: 0.9546874761581421

Best accuracy So Far: 0.9740625023841858
Total elapsed time: 00h 06m 03s

The hyperparameter search is complete. The optimal learning rate for the optimizer
is 0.001.
```

```
    model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=hp_learning_rate),
                  loss=tf.keras.losses.CategoricalCrossentropy(),
                  metrics=['accuracy'])

    return model
```

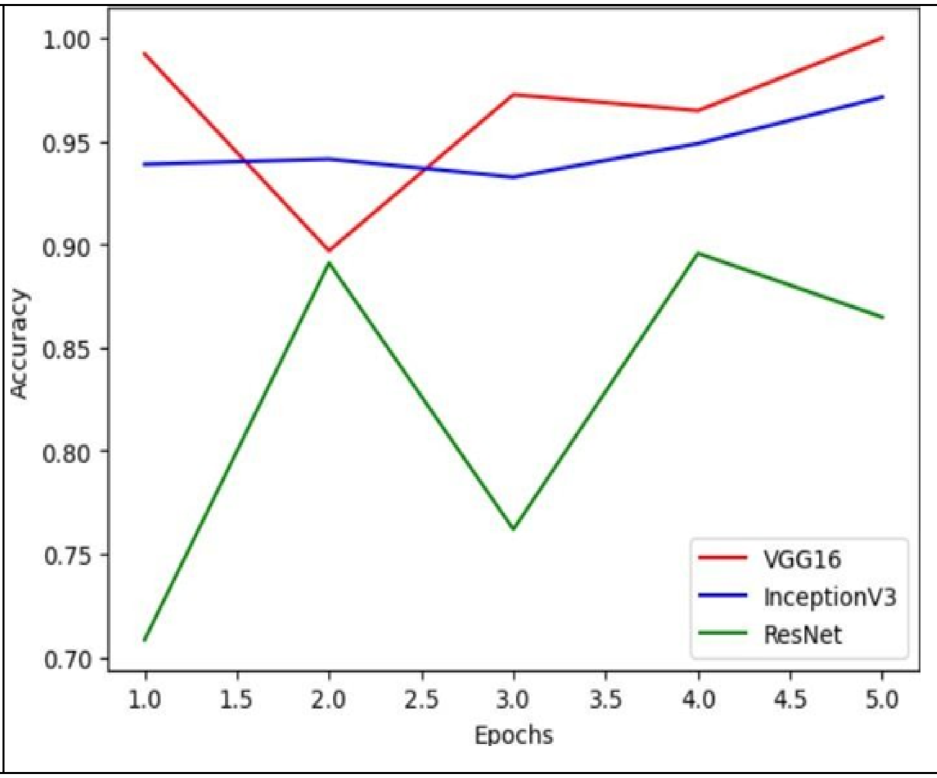
	Loss Function: loss='categorical_crossentropy' - Determines the errorbetween predicted and actualoutput for multi-class classification tasks.
	
	Optimizer: optimizer=Adam(learning_rate=0.01) **-** Adam optimizer witha learning rate of 0.01, adjusting the step size during training for better convergence.
	
	Metrics: metrics=['accuracy'] - Evaluation metric to measure the proportion of correctly classified examples out of the totalduring training andtesting.
	
	No. of epochs = 4
	
	`model = tuner.hypermodel.build(best_hps)`
	`history6 = model.fit(train, epochs=4)`
	
	`acc_per_epoch = history6.history['accuracy']`
	`best_epoch = acc_per_epoch.index(max(acc_per_epoch)) + 1`
	`print('Best epoch: %d' % (best_epoch,))`
	
	`Epoch 1/4`
	`160/160 [==============================] - 59s 365ms/step - loss: 0.2693 - accuracy: 0.9081`
	`Epoch 2/4`
	`160/160 [==============================] - 60s 372ms/step - loss: 0.0733 - accuracy: 0.9741`
	`Epoch 3/4`
	`160/160 [==============================] - 58s 359ms/step - loss: 0.0413 - accuracy: 0.9887`
	`Epoch 4/4`
	`160/160 [==============================] - 59s 368ms/step - loss: 0.0373 - accuracy: 0.9881`
	`Best epoch: 3`
	
	But we found best epochwas 3 so went with 3
	
	epochs.We found the accuracy to be 98.91 % ~ 99%

Resnet50	`resnet.compile(loss = "categorical_crossentropy", optimizer = "adam" , metrics = ["accuracy"])`
	
	`[ ] history3 = resnet.fit(train , validation_data = test , epochs = 5)`
	
	`Epoch 1/5`
	`160/160 [==============================] - 91s 528ms/step - loss: 1.6670 - accuracy: 0.5344 - val_loss: 1.5158 - val_accuracy: 0.7084`
	`Epoch 2/5`
	`160/160 [==============================] - 84s 526ms/step - loss: 0.8868 - accuracy: 0.6953 - val_loss: 0.2883 - val_accuracy: 0.8909`
	`Epoch 3/5`
	`160/160 [==============================] - 100s 627ms/step - loss: 0.5234 - accuracy: 0.7997 - val_loss: 0.7353 - val_accuracy: 0.7619`
	`Epoch 4/5`
	`160/160 [==============================] - 86s 535ms/step - loss: 0.5683 - accuracy: 0.7900 - val_loss: 0.2721 - val_accuracy: 0.8956`
	`Epoch 5/5`
	`160/160 [==============================] - 84s 526ms/step - loss: 0.6803 - accuracy: 0.7763 - val_loss: 0.3258 - val_accuracy: 0.8647`

	Loss Function: loss='categorical_crossentropy' - Determines the errorbetween predicted and actualoutput for multi-class classification tasks. **Optimizer: optimizer**=Adam - Adam optimizer with a learning rate of0.01,adjusting the step sizeduring training for better convergence. **Metrics: metrics=['accuracy']** - Evaluation metric to measure the proportion of correctly classified examples out of the totalduring training andtesting. For 5 epochs we found the best accuracy to be 77.63% ~ 78%
InceptionV3	`inception.compile(loss = "categorical_crossentropy", optimizer = "adam", metrics = ["accuracy"])` **Loss Function: loss='categorical_crossentropy'** - Determines the errorbetween predicted and actualoutput for multi-class classification tasks. **Optimizer: optimizer**=Adam - Adam optimizer with a learning rate of0.01,adjusting the step sizeduring training for better convergence. **Metrics: metrics=['accuracy']** - Evaluation metric to measure the proportion of correctly classified examples out of the totalduring training andtesting. For 5 epochs we found the best accuracy to be 97.63% ~ 98%

Final Model Selection Justification (2 Marks):

Final Model	Reasoning
VGG16	The decision to select VGG16 as the final optimized model for colon disease classification was based on its robust capability to extract relevant features from raw image data and its accuracy around 99%. We have built VGG16, INCEPTION V3 and RESNET50 out of which VGG16 had the highest accuracy. To refine its performance and ensure robustness, the VGG16 model underwent further optimization through techniques such as hyperparameter tuning. These enhancements were crucial in enhancing the model's accuracy, reliability, and generalizability for accurately diagnosing colon diseases from image data. Following is the graphwhich will helpvisualize the accuracies of the 3 models

Project Demo Video Link :
https://youtu.be/nmC6rgbvPOg?si=iyAVEHohZkUwIvcn