Final Project Report Template

1. Introduction
    1.1. Project overviews
    1.2. Objectives
2. Project Initialization and Planning Phase
    2.1. Define Problem Statement
    2.2. Project Proposal (Proposed Solution)
    2.3. Initial Project Planning
3. Data Collection and Preprocessing Phase
    3.1. Data Collection Plan and Raw Data Sources Identified
    3.2. Data Quality Report
    3.3. Data Preprocessing
4. Model Development Phase
    4.1. Model Selection Report
    4.2. Initial Model Training Code, Model Validation and Evaluation Report
5. Model Optimization and Tuning Phase
    5.1. Tuning Documentation
    5.2. Final Model Selection Justification
6. Results
    6.1. Output Screenshots
7. Advantages & Disadvantages
8. Conclusion
9. Future Scope
10. Appendix
    10.1 Source Code
    10.2 GitHub & Project Demo Link

Project Initialization and Planning Phase

Date	15 March 2024
Team ID	SWTID1720333657
Project Name	Wce Curated Colon Disease Classification
Maximum Marks	3 Marks

Define Problem Statements :

Current diagnostic methods for colon diseases, which heavily rely on physician interpretation of colonoscopy images, exhibit inconsistencies and inaccuracies. This can lead to missed diagnoses of critical conditions, particularly early-stage cancers, ultimately delaying treatment and potentially compromising patient outcomes. Additionally, limitations in diagnostic accuracy can necessitate unnecessary biopsies and procedures, causing discomfort for patients and incurring unnecessary costs for the healthcare system. Furthermore, the time-consuming nature of manual image analysis creates inefficiencies in clinical workflows, hindering timely decision-making and impacting the overall quality of patient care. Therefore, a novel and more robust approach is urgently needed to address these challenges. By harnessing the power of deep learning, we propose the development of a more accurate and efficient system for colon disease classification, with the ultimate goal of improving patient care, streamlining clinical workflows, and reducing healthcare costs.

Problem Statement (PS)	I am (Customer)	I'm trying to	But	Because	Which makes me feel
PS-1	a healthcare professional.	improve the accuracy and efficiency of my colonoscopy diagnoses.	the increasing number of colonoscopies I perform can be time-consuming.	the workload for gastroenterologists is growing, and time constraints can impact the thoroughness of examinations.	**empowered** to potentially improve the efficiency and accuracy of my colonoscopy diagnoses.
PS-2	patient with a family history of colon cancer.	catch any potential colon diseases early.	I'm anxious about the potential cost of a colonoscopy and any additional diagnostic procedures.	healthcare costs can be a significant burden, especially for those without comprehensive health insurance.	**hopeful** about the possibility of catching any potential colon diseases early at cheaper rates.

Project Initialization and Planning Phase

Date	15 March 2024
Team ID	SWTID1720333657
Project Title	Wce Curated Colon Disease Classification Using Deep
Maximum Marks	3 Marks

Project Proposal (Proposed Solution) template

This project proposes a novel solution to revolutionize colon disease diagnosis using deep learning. Current diagnostic methods based on colonoscopy images often lead to inconsistencies and missed early-stage cancers. Our solution aims to improve accuracy, efficiency, and patient outcomes by leveraging deep learning algorithms. Key features include automated image analysis for precise disease classification, reducing unnecessary biopsies and procedure costs. Personnel will include data scientists, medical experts, and software engineers. This initiative aims to streamline clinical workflows, enhance diagnostic precision, and ultimately improve patient care while optimizing healthcare resource utilization.

Project Overview	
Objective	The primary objective is to enhance colon disease diagnosis through the implementation of deep learning technology, aiming to improve diagnostic accuracy, streamline clinical workflows, and ultimately enhance patient care outcomes.
Scope	The project scope encompasses the development and implementation of a deep learning-based system for automated analysis of colonoscopy images. It focuses on improving diagnostic accuracy for colon diseases, particularly early-stage cancers, while streamlining clinical workflows to reduce unnecessary procedures and costs.
Problem Statement	
Description	Current colonoscopy diagnoses rely heavily on subjective interpretations of images by doctors, leading to inconsistencies and missed cancers.
Impact	Deep learning-aided colonoscopy diagnoses could means earlier cancer detection and better patient outcomes and fewer unnecessary procedures for patients and lower costs for healthcare.
Proposed Solution	
Approach	To tackle diagnosis inconsistencies, the project will leverage deep learning. We'll train a powerful image recognition model on a massive dataset of labeled colonoscopy images. This model will learn to identify disease signatures during training, ultimately assisting endoscopists in real-time during procedures.

Key Features	This solution goes beyond just automating analysis. It harnesses deep learning's power to identify subtle disease markers, potentially exceeding human accuracy and revolutionizing colon disease classification.

Resource Requirements

Resource Type	**Description**	**Specification/Allocation**
Hardware		
Computing Resources	CPU/GPU specifications, number of cores	AMD Ryzen 7 5700U with Radeon Graphics
Memory	RAM specifications	16 GB
Storage	Disk space for data, models, and logs	1 TB SSD
Software		
Frameworks	Python frameworks	Flask
Libraries	Additional libraries	Tensorflow, keras, numpy
Development Environment	IDE, version control	e.g., Jupyter Notebook, Git
Data		
Data	Source, size, format	Kaggle dataset, 4000 images

Data Collection and Preprocessing Phase

Date	15 March 2024
Team ID	SWTID1720333657
Project Title	Wce Curated Colon Disease Classification Using Deep
Maximum Marks	2 Marks

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 analysis and decision-making endeavor.

Data Collection Plan Template

Section	Description
Project Overview	Existing colonoscopy-based diagnostics, dependent on physician interpretation, exhibit limitations in accuracy and consistency. This can lead to missed diagnoses, unnecessary procedures, and workflow inefficiencies. We propose a novel deep learning system for colon disease classification to address these challenges, aiming to enhance diagnostic precision, streamline clinical workflows, and ultimately improve patient care and cost-effectiveness.
Data Collection Plan	· Existing **Datasets:** Look for Kaggle datasets with colonoscopy-related images (e.g., histology) or anonymized data (e.g., polyp labels). · Partnerships**:** Collaborate with medical institutions for access to anonymized data or synthetic images. · Public **Repositories:** Explore resources like TCIA for potentially relevant datasets.

Raw Data Sources Identified	· Kaggle **Datasets:** Look for anonymized colonoscopy data (e.g., polyp labels) or related images (e.g., histology). · Medical **Collaborations:** Partner with hospitals for access to anonymized colonoscopy data. (Benefits: relevant data, diverse cases. Challenges: approvals, anonymization) · Public **Repositories:** Explore TCIA ([Cancer Imaging Archive]) or NIH trials for potentially relevant datasets, keeping in mind data usage licenses and ethics.

Raw Data Sources Template

Source Name	**Description**	**Location/URL**	**Format**	**Size**	**Access Permissions**

Kaggle Dataset	Data is comprised of colon disease images (Normal, Ulcerative Colitis, Polyps) from Kaggle.	https://www.kaggle.com/datasets/francismon/curated-colon-dataset-for-deep-learning/data	Image	2 GB	Public

Data Collection and Preprocessing Phase

Date	15 March 2024
Team ID	SWTID1720333657
Project Title	Wce Curated Colon Disease Classification Using Deep
Maximum Marks	2 Marks

Data Quality Report Template

The data originated from a Kaggle dataset, which generally implies a level of usability. However, we identified opportunities to further enhance the data quality. We employed a technique called data augmentation, which essentially creates additional data points from existing ones. This approach effectively bolsters the dataset's robustness without requiring substantial modifications to the original information.

Data Source	Data Quality Issue	Severity	Resolution Plan
Kaggle dataset	No issues faced. Just had to ameliorate the quality of raw data.	Low	With help of data augmentation.
…	…	…	…

Data Collection and Preprocessing Phase

Date	15 March 2024
Team ID	SWTID1720333657
Project Title	Wce Curated Colon Disease Classification Using Deep
Maximum Marks	6 Marks

Preprocessing Template

The images will be preprocessed by resizing, normalizing, augmenting, denoising, adjusting contrast, detecting edges, converting color space, cropping, batch normalizing, and whitening data. These steps will enhance data quality, promote model generalization, and improve convergence during neural network training, ensuring robust and efficient performance across various computer vision tasks.

Section	**Description**
Data Overview	Data is comprised of colon disease images (Normal, Ulcerative Colitis, Polyps) from Kaggle.
Resizing	Train: 3200 belonging to 4 classes Test: 800 belonging to 4 classes
Normalization	1./255
Data Augmentation	Rotation range=40 width shift range=0.2 height shift range=0.2 shear range=0.2 zoom range=0.2 horizontal flip=True
Denoising	Applied denoising filters to reduce noise in the images
Edge Detection	-
Color Space Conversion	-
Image Cropping	Resize image to 224 x 224
Batch Normalization	-
Data Preprocessing Code Screenshots	

<table>
<tr><td>Loading Data</td><td><pre>train_datagen = ImageDataGenerator(rescale=1./255,
    rotation_range=40,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    fill_mode='nearest'

)</pre>
<pre>train_data = train_datagen.flow_from_directory('C:/Users/DNZN/Desktop/machine learning tutorial/train',target_size = (224,224),batch_size = 32,class_mode</pre>
<pre>Found 3200 images belonging to 4 classes.</pre>
<pre>test_data = train_datagen.flow_from_directory('C:/Users/DNZN/Desktop/machine learning tutorial/test',target_size = (224,224),batch_size = 32,class_mode =</pre>
<pre>Found 800 images belonging to 4 classes.</pre></td></tr>
<tr><td>Resizing</td><td>

</td></tr>
<tr><td>Normalization</td><td><pre>train_datagen = ImageDataGenerator(rescale=1./255,
    rotation_range=40,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    fill_mode='nearest'

)
train_datagen</pre>
<pre><keras.src.legacy.preprocessing.image.ImageDataGenerator at 0x28252cc1520></pre></td></tr>
<tr><td>Data Augmentation</td><td><pre>train_datagen = ImageDataGenerator(rescale=1./255,
    rotation_range=40,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    fill_mode='nearest'

)
train_datagen</pre>
<pre><keras.src.legacy.preprocessing.image.ImageDataGenerator at 0x28252cc1520></pre></td></tr>
<tr><td>Denoising</td><td><pre>def denoise_image(img):
  # Convert to grayscale (assuming grayscale is suitable for denoising)
  gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

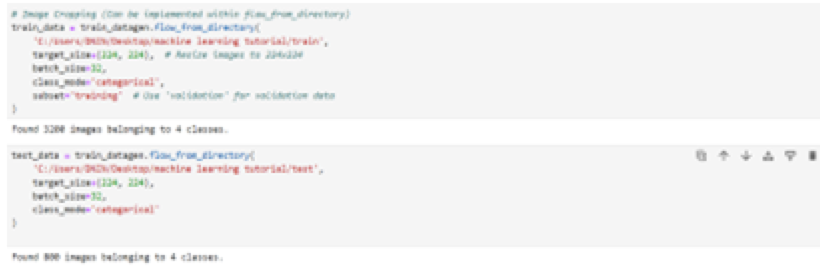
  # Apply non-local means denoising (replace with your preferred filter)
  denoised_img = cv2.fastNlMeansDenoising(gray, None, 10, 10, 7, 21)

  return denoised_img
train_datagen.preprocessing_function = denoise_image
train_datagen = ImageDataGenerator(
    rescale=1./255,
    rotation_range=40,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    fill_mode='nearest'
)

# Add denoising to the data augmentation pipeline
train_datagen.preprocessing_function = denoise_image
train_datagen = ImageDataGenerator(rescale=1./255,
    rotation_range=40,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    fill_mode='nearest'

)</pre></td></tr>
<tr><td>Edge Detection</td><td>-</td></tr>
</table>

Color Space Conversion	-

Image Cropping	
	# Image Cropping (Can be implemented within flow_from_directory)
	train_data = train_datagen.flow_from_directory(
	'C:/Users/[illegible]/Desktop/machine learning tutorial/train',
	target_size=(224, 224), # Resize images to 224x224
	batch_size=32,
	class_mode='categorical',
	subset='training' # Use 'validation' for validation data
	)
	Found 3200 images belonging to 4 classes.
	test_data = train_datagen.flow_from_directory(
	'C:/Users/[illegible]/Desktop/machine learning tutorial/test',
	target_size=(224, 224),
	batch_size=32,
	class_mode='categorical'
	)
	Found 800 images belonging to 4 classes.

Batch Normalization	-

Model Development Phase Template

Date	15 March 2024
Team ID	SWTID1720333657
Project Title	Wce Curated Colon Disease Classification Using Deep
Maximum Marks	5 Marks

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 suitable model for the task at hand.

Model Selection Report:

Model	Description

VGG16	VGG16 is a deep convolutional neural network (CNN) architecture renowned for its effectiveness in image recognition. It employs a repetitive stacking of small convolutional filters with rectified linear units (ReLUs), gradually extracting increasingly intricate features from images. This architecture, while not the state-of-the-art, played a pivotal role in advancing computer vision research due to its clear design and strong performance. VGG16 serves as a foundation for many contemporary image classification and object detection tasks.
ResNet-50	ResNet50 is a deep convolutional neural network architecture known for its image recognition capabilities. It utilizes 50 residual blocks, a clever design that allows the network to learn from past layers and avoid vanishing gradients - a common challenge in deep learning. This "shortcut" learning enables ResNet50 to achieve high accuracy while maintaining a complex structure. Often pre-trained on massive datasets, ResNet50 serves as a powerful foundation for fine-tuning in specific computer vision tasks like object detection or image classification.
EfficientNet	EfficientNet is a cutting-edge convolutional neural network architecture designed for optimal performance in image recognition. It achieves state-of-the-art accuracy while maintaining computational efficiency. Unlike traditional models, EfficientNet utilizes a compound scaling method, dynamically adjusting depth, width, and resolution for a desired accuracy-efficiency trade-off. This offers a family of models (B0-B7) suitable for various computing resources. B0 prioritizes speed, while B7 maximizes accuracy. This versatility makes EfficientNet ideal for tasks like object detection and image classification on diverse devices

Model Development Phase Template

Date	15 March 2024
Team ID	SWTID1720333657
Project Title	Wce Curated Colon Disease Classification Using Deep

Maximum Marks	10 Marks

Initial Model Training Code, 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 summary and training and validation performance metrics for multiple models, presented through respective screenshots.

Initial Model Training Code (5 marks):

Data Augmentation:

```
[3]: from tensorflow.keras.preprocessing.image import ImageDataGenerator

[4]: train_datagen = ImageDataGenerator(
         rotation_range=40,
         width_shift_range=0.2,
         height_shift_range=0.2,
         shear_range=0.2,
         zoom_range=0.2,
         horizontal_flip=True,
         fill_mode='nearest'
     )

[5]: train_data = train_datagen.flow_from_directory('C:/Users/DNIN/Desktop/machine learning tutorial/train',target_size = (224,224),batch_size = 32,class_mode

     Found 3200 images belonging to 4 classes.

[6]: test_data = train_datagen.flow_from_directory('C:/Users/DNIN/Desktop/machine learning tutorial/test',target_size = (224,224),batch_size = 32,class_mode =

     Found 800 images belonging to 4 classes.
```

VGG16 Model:

```
[1]: import tensorflow as tf
     from tensorflow import keras
     from tensorflow.keras.applications.vgg16 import VGG16
     from tensorflow.keras.layers import Flatten
     from tensorflow.keras.models import Model
     from tensorflow.keras.layers import Dense
     from tensorflow.keras.activations import softmax
     from keras.api import activations

[2]: Image_size = [224,224]
     sol = VGG16(input_shape = Image_size + [3] ,include_top = False)
     for i in sol.layers:
         i.trainable = False
     y = Flatten()(sol.output)

[3]: final = Dense(4, activation = 'softmax')(y)
     vgg16_model = Model(inputs=sol.input, outputs=final)

11]: vgg16_model.summary()
```

```
# Assuming L2 weight of 0.01
loss_weight = 0.05

vgg16_model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'], loss_weights=loss_weight)
```

```
from tensorflow.keras.callbacks import EarlyStopping
# Set up early stopping to monitor validation accuracy
early_stopping = EarlyStopping(monitor='val_accuracy', patience=3, mode = 'max')
vgg16_model.fit(train_data, epochs = 15, validation_data = test_data,  callbacks = [early_stopping])
```

```
[38]: vgg16_model.save('cnn.h5')
```

```
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format is considered legacy. We recommend using instead the native Keras format, e.g. `model.save('my_model.keras')` or `keras.saving.save_model(model, 'my_model.keras')`.
```

```
[52]: from keras.preprocessing import image
from keras.applications.vgg16 import preprocess_input
from tensorflow.keras.preprocessing.image import load_img, img_to_array
import numpy as np
```

```
[53]: labels = ['0_normal','1_ulcerative_colitis','2_polyps','3_esophagitis']
img_path = 'C:/Users/DNIN/Desktop/machine learning tutorial/test/1_ulcerative_colitis/test_ulcer_ (164).jpg'
#"C:\Users\DNIN\Desktop\machine Learning tutorial\test\1_ulcerative_colitis\test_ulcer_ (164).jpg"
img = load_img(img_path, target_size=(224,224))
x = img_to_array(img)
x = preprocess_input(x)
preds = vgg16_model.predict(np.array([x]))
preds
```

ResNet-50 Model:

```
[4]: import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.applications.resnet50 import ResNet50
from tensorflow.keras.layers import Flatten
from tensorflow.keras.models import Model
```

```
[6]: Image_size = [224, 224]

# Load the pre-trained ResNet-50 model
resnet50 = ResNet50(input_shape=Image_size + [3], include_top=False)

# Freeze all layers to prevent training
for layer in resnet50.layers:
    layer.trainable = False

# Flatten the output from ResNet-50
y = tf.keras.layers.Flatten()(resnet50.output)
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/resnet/resnet50_weights_tf_dim_ordering_tf_kernels_notop.h5
94765736/94765736 ━━━━━━━━━━━━━━━━━━━━ 17s 0us/step
```

```
[16]: # Assuming L2 weight of 0.01
loss_weight = 0.01

resnet50_model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'], loss_weights=loss_weight)
```

```
[21]: from tensorflow.keras.callbacks import EarlyStopping

# Set up early stopping to monitor validation accuracy
early_stopping = EarlyStopping(monitor='val_accuracy', patience=3)
# Train the model with early stopping
resnet50_model.fit(train_data, epochs=20, validation_data=test_data, callbacks=[early_stopping])
```

EfficientNet Model:

```
[7]: from tensorflow.keras.applications import EfficientNetB0
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.layers import Flatten
from tensorflow.keras.models import Model
```

```
[8]:
Image_size = [224, 224]

efficientnet_model = EfficientNetB0(weights='imagenet', include_top=False, input_shape=Image_size + [3])
for layer in efficientnet_model.layers:
  layer.trainable = False
```

```
Downloading data from https://storage.googleapis.com/keras-applications/efficientnetb0_notop.h5
16705208/16705208 ━━━━━━━━━━━━━━━━━━━━ 4s 0us/step
```

```
[7]: from tensorflow.keras.layers import Dense
     from tensorflow.keras.activations import softmax
     from keras.api import activations
     y = tf.keras.layers.Flatten()(efficientnet_model.output)

     num_classes = 4  # Adjust based on your dataset

     final_layer = Dense(num_classes, activation='softmax')(y)

     efficientnet_model = Model(inputs=efficientnet_model.input, outputs=final_layer)
```

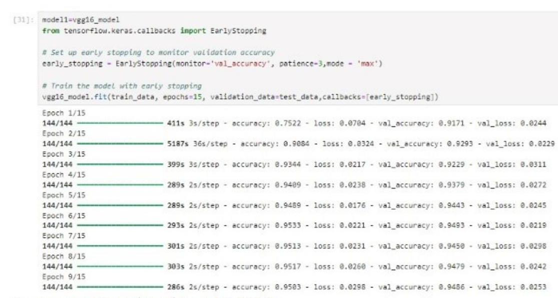
```
[10]: efficientnet_model.summary()
```

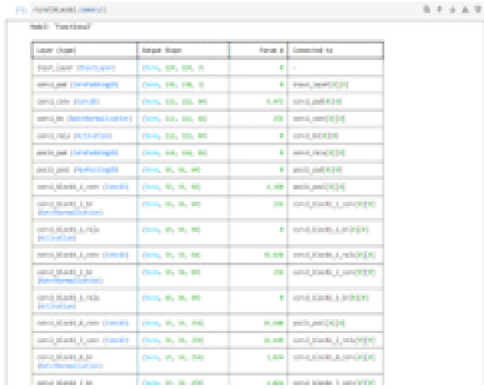
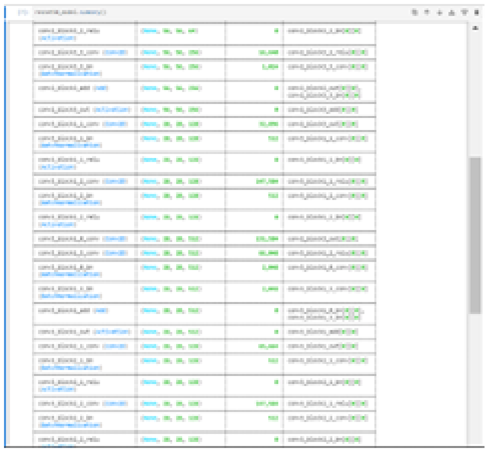
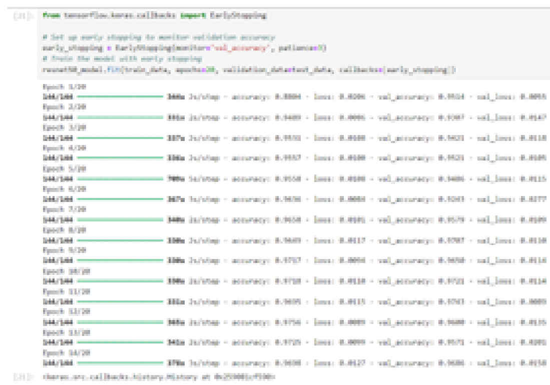
```
loss_weight = 0.05
efficientnet_model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'],loss_weights = loss_weight)
```

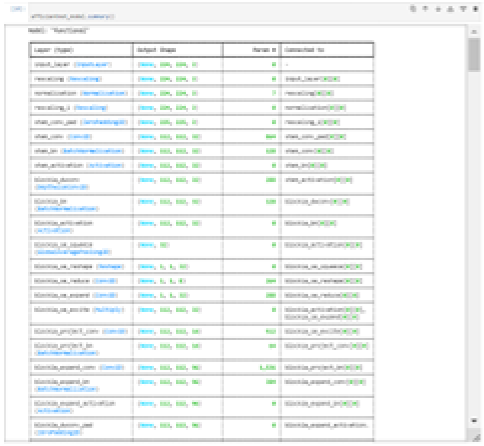
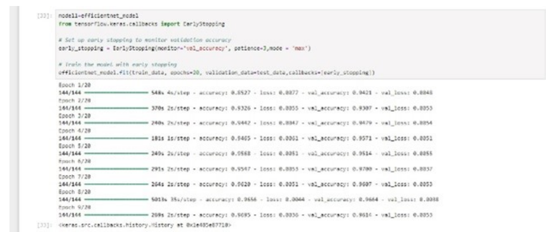
```
from tensorflow.keras.callbacks import EarlyStopping

# Set up early stopping to monitor validation accuracy
early_stopping = EarlyStopping(monitor='val_accuracy', patience=3,mode = 'max')

# Train the model with early stopping
efficientnet_model.fit(train_data, epochs=20, validation_data = test_data , callbacks=[early_stopping])
```

Model Validation and Evaluation Report (5 marks):

Model	Summary	Training and Validation Performance Metrics

VGG16
Model
		
		[31]: model1=vgg16_model
		from tensorflow.keras.callbacks import EarlyStopping
		
		# Set up early stopping to monitor validation accuracy
		early_stopping = EarlyStopping(monitor='val_accuracy', patience=3,mode = 'max')
		
		# Train the model with early stopping
		vgg16_model.fit(train_data, epochs=15, validation_data=test_data,callbacks=[early_stopping])
		Epoch 1/15
		144/144 ━━━━━━━━━━ 411s 3s/step - accuracy: 0.7522 - loss: 0.0704 - val_accuracy: 0.9171 - val_loss: 0.0244
		Epoch 2/15
		144/144 ━━━━━━━━━━ 5187s 36s/step - accuracy: 0.9084 - loss: 0.0324 - val_accuracy: 0.9293 - val_loss: 0.0229
		Epoch 3/15
		144/144 ━━━━━━━━━━ 399s 3s/step - accuracy: 0.9344 - loss: 0.0217 - val_accuracy: 0.9229 - val_loss: 0.0311
		Epoch 4/15
		144/144 ━━━━━━━━━━ 289s 2s/step - accuracy: 0.9409 - loss: 0.0238 - val_accuracy: 0.9379 - val_loss: 0.0272
		Epoch 5/15
		144/144 ━━━━━━━━━━ 289s 2s/step - accuracy: 0.9489 - loss: 0.0176 - val_accuracy: 0.9443 - val_loss: 0.0245
		Epoch 6/15
		144/144 ━━━━━━━━━━ 293s 2s/step - accuracy: 0.9533 - loss: 0.0221 - val_accuracy: 0.9493 - val_loss: 0.0219
		Epoch 7/15
		144/144 ━━━━━━━━━━ 301s 2s/step - accuracy: 0.9513 - loss: 0.0231 - val_accuracy: 0.9450 - val_loss: 0.0298
		Epoch 8/15
		144/144 ━━━━━━━━━━ 303s 2s/step - accuracy: 0.9517 - loss: 0.0260 - val_accuracy: 0.9479 - val_loss: 0.0242
		Epoch 9/15
		144/144 ━━━━━━━━━━ 286s 2s/step - accuracy: 0.9503 - loss: 0.0298 - val_accuracy: 0.9486 - val_loss: 0.0253
		[31]: <keras.src.callbacks.history.History at 0x24b9f1369f0>

Resnet-50	  	 from tensorflow.keras.callbacks import EarlyStopping

EfficientNet	  	

Model Development Phase Template

Date	15 March 2024

Team ID	SWTID1720333657
Project Title	Wce Curated Colon Disease Classification Using Deep
Maximum Marks	5 Marks

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 suitable model for the task at hand.

Model Selection Report:

Model	Description
VGG16	VGG16 is a deep convolutional neural network (CNN) architecture renowned for its effectiveness in image recognition. It employs a repetitive stacking of small convolutional filters with rectified linear units (ReLUs), gradually extracting increasingly intricate features from images. This architecture, while not the state-of-the-art, played a pivotal role in advancing computer vision research due to its clear design and strong performance. VGG16 serves as a foundation for many contemporary image classification and object detection tasks.
ResNet-50	ResNet50 is a deep convolutional neural network architecture known for its image recognition capabilities. It utilizes 50 residual blocks, a clever design that allows the network to learn from past layers and avoid vanishing gradients - a common challenge in deep learning. This "shortcut" learning enables ResNet50 to achieve high accuracy while maintaining a complex structure. Often pre-trained on massive datasets, ResNet50 serves as a powerful foundation for fine-tuning in specific computer vision tasks like object detection or image classification.

EfficientNet	EfficientNet is a cutting-edge convolutional neural network architecture designed for optimal performance in image recognition. It achieves state-of-the-art accuracy while maintaining computational efficiency. Unlike traditional models, EfficientNet utilizes a compound scaling method, dynamically adjusting depth, width, and resolution for a desired accuracy-efficiency trade-off. This offers a family of models (B0-B7) suitable for various computing resources. B0 prioritizes speed, while B7 maximizes accuracy. This versatility makes EfficientNet ideal for tasks like object detection and image classification on diverse devices

Model Optimization and Tuning Phase Template

Date	15 March 2024
Team ID	SWTID1720333657
Project Title	Wce Curated Colon Disease Classification Using Deep
Maximum Marks	10 Marks

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 accuracy and efficiency.

Hyperparameter Tuning Documentation (8 Marks):

Model	Tuned Hyperparameters

<table>
<tr>
<td>VGG16</td>
<td>
· Loss ('categorical_crossentropy'): Measures model performance, lower is better.
· Metrics (['accuracy']): Tracks training progress (percentage of correct predictions).
· Optimizer ('adam'): Guides weight updates during training.
· Epochs (15): Maximum number of training iterations.
· Early Stopping: Stops training if validation accuracy doesn't improve for 3 epochs (prevents overfitting).
<pre><code># Assuming L2 weight of 0.01
loss_weight = 0.05

vgg16_model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'], loss_weights=loss_weight)</code></pre>
<pre><code>from tensorflow.keras.callbacks import EarlyStopping
# Set up early stopping to monitor validation accuracy
early_stopping = EarlyStopping(monitor='val_accuracy', patience=3, mode = 'max')
vgg16_model.fit(train_data, epochs = 15, validation_data = test_data, callbacks = [early_stopping])</code></pre>
</td>
</tr>
<tr>
<td>ResNet-50</td>
<td>
· Loss ('categorical_crossentropy'): Measures model performance, lower is better.
· Metrics (['accuracy']): Tracks training progress (percentage of correct predictions).
· Optimizer ('adam'): Guides weight updates during training.
· Epochs (20): Maximum number of training iterations.
· Early Stopping: Stops training if validation accuracy doesn't improve for 3 epochs (prevents overfitting).
<pre><code># Assuming L2 weight of 0.01
loss_weight = 0.01

resnet50_model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'], loss_weights=loss_weight)</code></pre>
<pre><code>from tensorflow.keras.callbacks import EarlyStopping

# Set up early stopping to monitor validation accuracy
early_stopping = EarlyStopping(monitor='val_accuracy', patience=3)
# Train the model with early stopping
resnet50_model.fit(train_data, epochs=20, validation_data=test_data, callbacks=[early_stopping])</code></pre>
</td>
</tr>
</table>

<table>
<tr>
<td>EfficentNet</td>
<td>

· Loss **('categorical_crossentropy')**: Measures model performance, lower is better.
· Metrics **(['accuracy'])**: Tracks training progress (percentage of correct predictions).
· Optimizer **('adam')**: Guides weight updates during training.
· Epochs **(20)**: Maximum number of training iterations.
· Early **Stopping**: Stops training if validation accuracy doesn't improve for 3 epochs (prevents overfitting).

```
loss_weight = 0.05
efficientnet_model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'],loss_weights = loss_weight)
```

```
from tensorflow.keras.callbacks import EarlyStopping

# Set up early stopping to monitor validation accuracy
early_stopping = EarlyStopping(monitor='val_accuracy', patience=3,mode = 'max')

# Train the model with early stopping
efficientnet_model.fit(train_data, epochs=20, validation_data = test_data , callbacks=[early_stopping])
```

</td>
</tr>
</table>

Final Model Selection Justification (2 Marks):

Final Model	Reasoning

EfficientNet	Based on the metrics provided, both ResNet-50 and EfficientNet models exhibit high performance, but each has its strengths. ResNet-50 achieves a final epoch accuracy of 0.9698 and a validation accuracy of 0.9686, slightly surpassing EfficientNet's final epoch accuracy of 0.9695 and validation accuracy of 0.9614. However, EfficientNet demonstrates a significantly lower validation loss of 0.0053 compared to ResNet-50's 0.0158, indicating better performance in minimizing error on the validation set. Given this lower validation loss, EfficientNet appears to generalize better and might perform more reliably on unseen data. Thus, despite ResNet-50's marginally higher accuracy, EfficientNet's superior validation loss suggests it is the preferable model for achieving better overall performance and generalization.