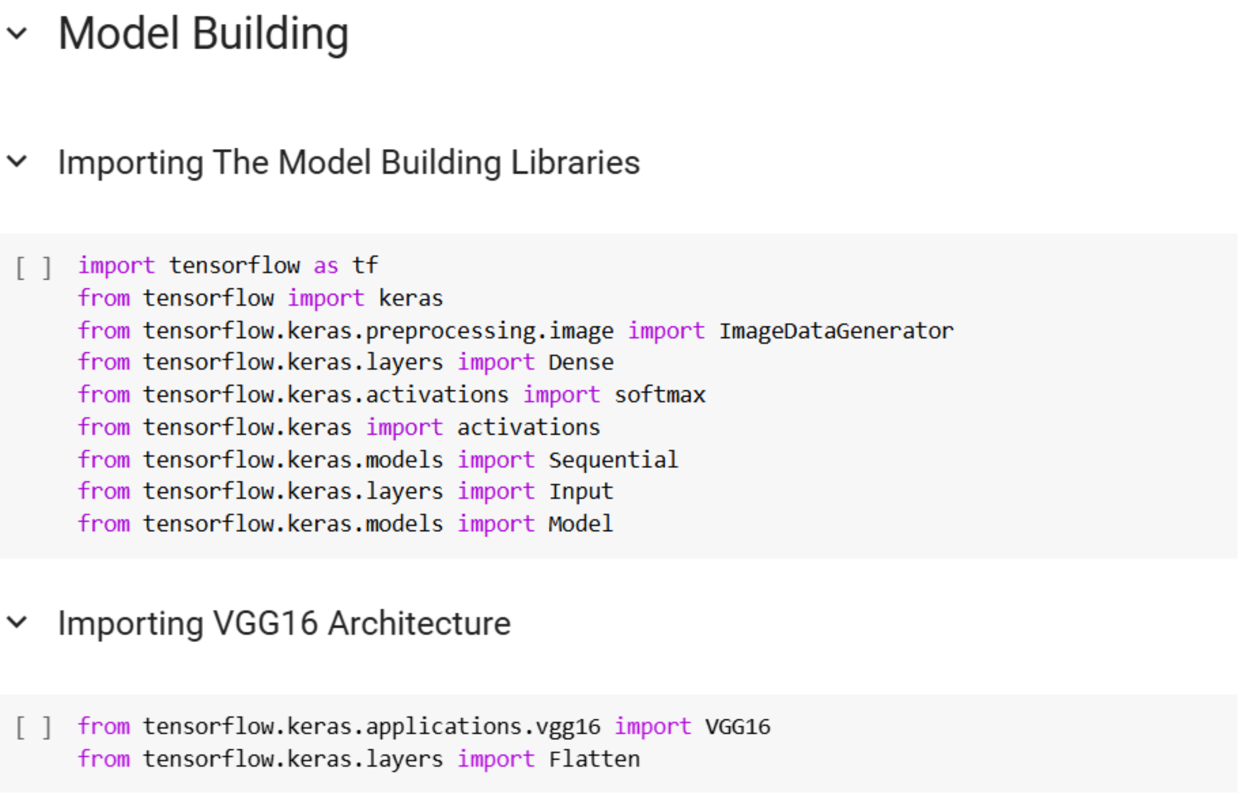
Model Development Phase Template

Date	15 March 2024
Team ID	SWTID1719937289
Project Title	WCE Curated Colon Disease Classification using Deep Learning
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 model, presented through respective screenshots.

Initial Model Training Code (5 marks):



Model Building

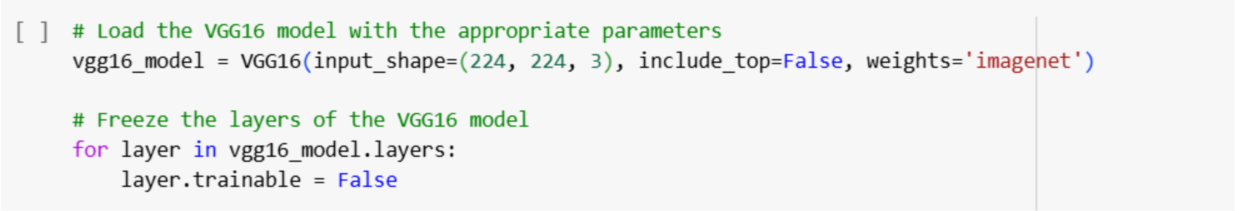
Importing The Model Building Libraries

```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.layers import Dense
from tensorflow.keras.activations import softmax
from tensorflow.keras import activations
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Input
from tensorflow.keras.models import Model
```

Importing VGG16 Architecture

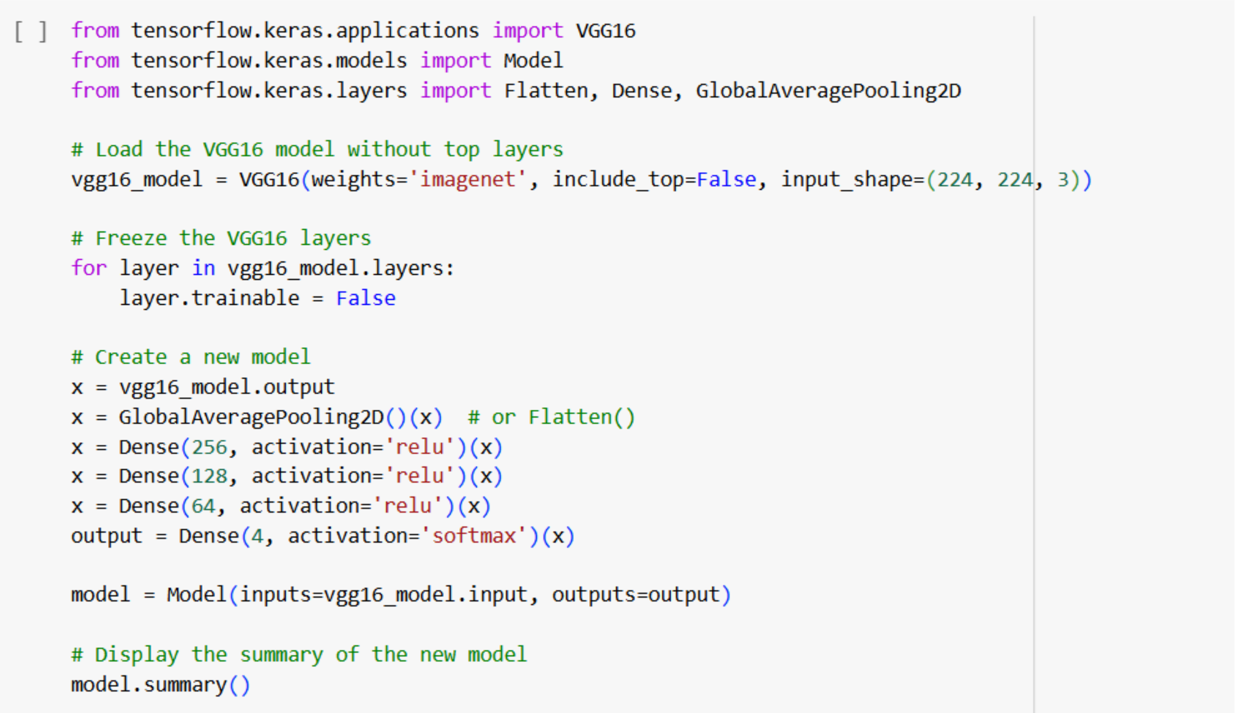
```
from tensorflow.keras.applications.vgg16 import VGG16
from tensorflow.keras.layers import Flatten
```

Initializing the Model



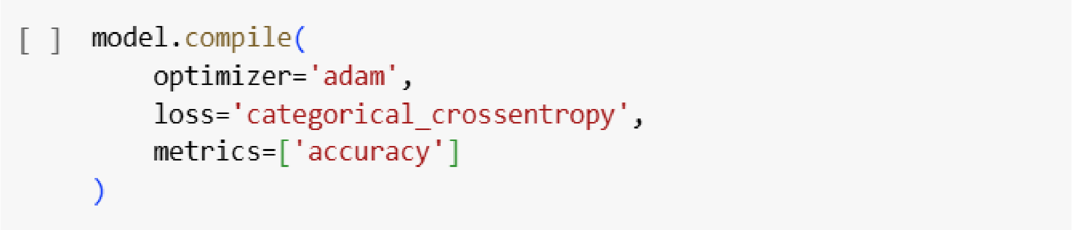
```
# Load the VGG16 model with the appropriate parameters
vgg16_model = VGG16(input_shape=(224, 224, 3), include_top=False, weights='imagenet')

# Freeze the layers of the VGG16 model
for layer in vgg16_model.layers:
    layer.trainable = False
```

Adding Fully Connected Layer



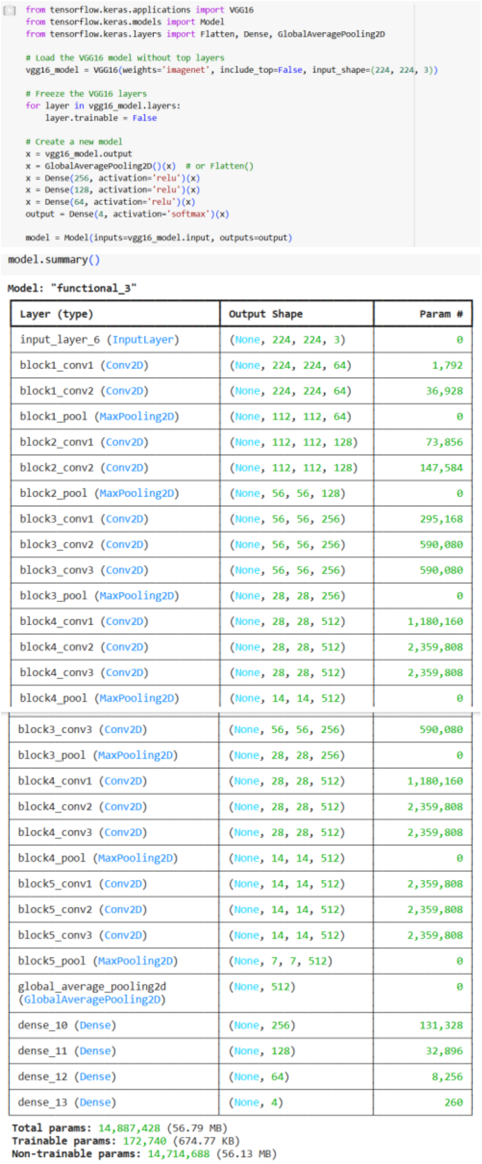
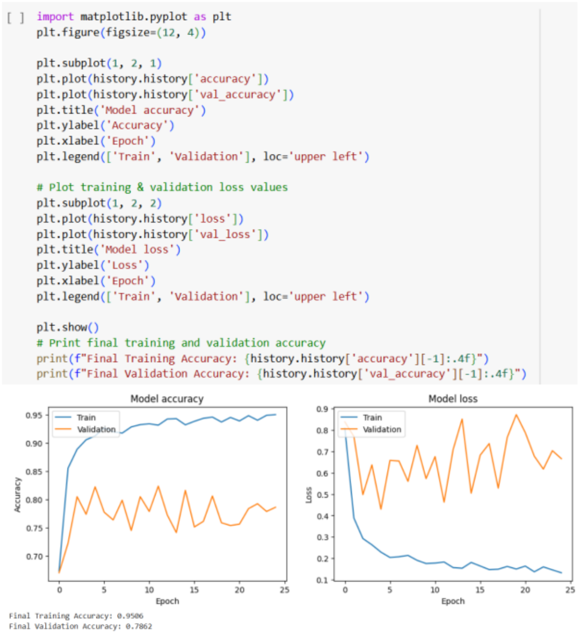
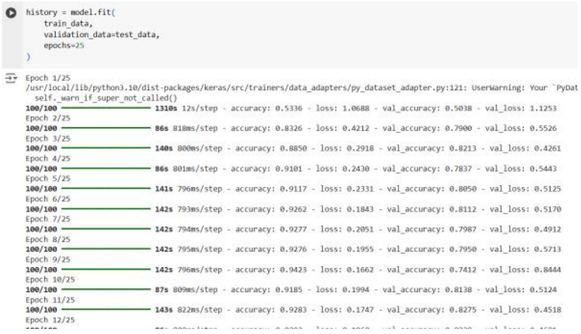
```
from tensorflow.keras.applications import VGG16
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Flatten, Dense, GlobalAveragePooling2D

# Load the VGG16 model without top layers
vgg16_model = VGG16(weights='imagenet', include_top=False, input_shape=(224, 224, 3))

# Freeze the VGG16 layers
for layer in vgg16_model.layers:
    layer.trainable = False

# Create a new model
x = vgg16_model.output
x = GlobalAveragePooling2D()(x)  # or Flatten()
x = Dense(256, activation='relu')(x)
x = Dense(128, activation='relu')(x)
x = Dense(64, activation='relu')(x)
output = Dense(4, activation='softmax')(x)

model = Model(inputs=vgg16_model.input, outputs=output)

# Display the summary of the new model
model.summary()
```

Configure the Learning Process



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

Model Validation and Evaluation Report (5 marks):

Model	Summary	Training and Validation Performance Metrics
VGG16		 

```python
from tensorflow.keras.applications import VGG16
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Flatten, Dense, GlobalAveragePooling2D

# Load the VGG16 model without top layers
vgg16_model = VGG16(weights='imagenet', include_top=False, input_shape=(224, 224, 3))

# Freeze the VGG16 layers
for layer in vgg16_model.layers:
    layer.trainable = False

# Create a new model
x = vgg16_model.output
x = GlobalAveragePooling2D()(x)  # or Flatten()
x = Dense(256, activation='relu')(x)
x = Dense(128, activation='relu')(x)
x = Dense(64, activation='relu')(x)
output = Dense(4, activation='softmax')(x)

model = Model(inputs=vgg16_model.input, outputs=output)
```

```python
model.summary()
```

Model: "functional_3"

Layer (type)	Output Shape	Param #
input_layer_6 (InputLayer)	(None, 224, 224, 3)	0
block1_conv1 (Conv2D)	(None, 224, 224, 64)	1,792
block1_conv2 (Conv2D)	(None, 224, 224, 64)	36,928
block1_pool (MaxPooling2D)	(None, 112, 112, 64)	0
block2_conv1 (Conv2D)	(None, 112, 112, 128)	73,856
block2_conv2 (Conv2D)	(None, 112, 112, 128)	147,584
block2_pool (MaxPooling2D)	(None, 56, 56, 128)	0
block3_conv1 (Conv2D)	(None, 56, 56, 256)	295,168
block3_conv2 (Conv2D)	(None, 56, 56, 256)	590,080
block3_conv3 (Conv2D)	(None, 56, 56, 256)	590,080
block3_pool (MaxPooling2D)	(None, 28, 28, 256)	0
block4_conv1 (Conv2D)	(None, 28, 28, 512)	1,180,160
block4_conv2 (Conv2D)	(None, 28, 28, 512)	2,359,808
block4_conv3 (Conv2D)	(None, 28, 28, 512)	2,359,808
block4_pool (MaxPooling2D)	(None, 14, 14, 512)	0
block3_conv3 (Conv2D)	(None, 56, 56, 256)	590,080
block3_pool (MaxPooling2D)	(None, 28, 28, 256)	0
block4_conv1 (Conv2D)	(None, 28, 28, 512)	1,180,160
block4_conv2 (Conv2D)	(None, 28, 28, 512)	2,359,808
block4_conv3 (Conv2D)	(None, 28, 28, 512)	2,359,808
block4_pool (MaxPooling2D)	(None, 14, 14, 512)	0
block5_conv1 (Conv2D)	(None, 14, 14, 512)	2,359,808
block5_conv2 (Conv2D)	(None, 14, 14, 512)	2,359,808
block5_conv3 (Conv2D)	(None, 14, 14, 512)	2,359,808
block5_pool (MaxPooling2D)	(None, 7, 7, 512)	0
global_average_pooling2d (GlobalAveragePooling2D)	(None, 512)	0
dense_10 (Dense)	(None, 256)	131,328
dense_11 (Dense)	(None, 128)	32,896
dense_12 (Dense)	(None, 64)	8,256
dense_13 (Dense)	(None, 4)	260

```
Total params: 14,887,428 (56.79 MB)
Trainable params: 172,740 (674.77 KB)
Non-trainable params: 14,714,688 (56.13 MB)
```

```python
import matplotlib.pyplot as plt
plt.figure(figsize=(12, 4))

plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Model accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='upper left')

# Plot training & validation loss values
plt.subplot(1, 2, 2)
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='upper left')

plt.show()
# Print final training and validation accuracy
print(f"Final Training Accuracy: {history.history['accuracy'][-1]:.4f}")
print(f"Final Validation Accuracy: {history.history['val_accuracy'][-1]:.4f}")
```

```
Final Training Accuracy: 0.9506
Final Validation Accuracy: 0.7862
```

```python
history = model.fit(
    train_data,
    validation_data=test_data,
    epochs=25
)
```

```
Epoch 1/25
/usr/local/lib/python3.10/dist-packages/keras/src/trainers/data_adapters/py_dataset_adapter.py:121: UserWarning: Your `PyDat
  self._warn_if_super_not_called()
100/100 ━━━━━━━━━━━━━━━━━━━━ 1310s 12s/step - accuracy: 0.5336 - loss: 1.0688 - val_accuracy: 0.5038 - val_loss: 1.1253
Epoch 2/25
100/100 ━━━━━━━━━━━━━━━━━━━━ 86s 818ms/step - accuracy: 0.8326 - loss: 0.4212 - val_accuracy: 0.7900 - val_loss: 0.5526
Epoch 3/25
100/100 ━━━━━━━━━━━━━━━━━━━━ 140s 800ms/step - accuracy: 0.8850 - loss: 0.2918 - val_accuracy: 0.8213 - val_loss: 0.4261
Epoch 4/25
100/100 ━━━━━━━━━━━━━━━━━━━━ 86s 801ms/step - accuracy: 0.9101 - loss: 0.2430 - val_accuracy: 0.7837 - val_loss: 0.5443
Epoch 5/25
100/100 ━━━━━━━━━━━━━━━━━━━━ 141s 796ms/step - accuracy: 0.9117 - loss: 0.2331 - val_accuracy: 0.8050 - val_loss: 0.5125
Epoch 6/25
100/100 ━━━━━━━━━━━━━━━━━━━━ 142s 793ms/step - accuracy: 0.9262 - loss: 0.1843 - val_accuracy: 0.8112 - val_loss: 0.5170
Epoch 7/25
100/100 ━━━━━━━━━━━━━━━━━━━━ 142s 794ms/step - accuracy: 0.9277 - loss: 0.2051 - val_accuracy: 0.7987 - val_loss: 0.4912
Epoch 8/25
100/100 ━━━━━━━━━━━━━━━━━━━━ 142s 795ms/step - accuracy: 0.9276 - loss: 0.1955 - val_accuracy: 0.7950 - val_loss: 0.5713
Epoch 9/25
100/100 ━━━━━━━━━━━━━━━━━━━━ 142s 796ms/step - accuracy: 0.9423 - loss: 0.1662 - val_accuracy: 0.7412 - val_loss: 0.8444
Epoch 10/25
100/100 ━━━━━━━━━━━━━━━━━━━━ 87s 809ms/step - accuracy: 0.9185 - loss: 0.1994 - val_accuracy: 0.8138 - val_loss: 0.5124
Epoch 11/25
100/100 ━━━━━━━━━━━━━━━━━━━━ 143s 822ms/step - accuracy: 0.9283 - loss: 0.1747 - val_accuracy: 0.8275 - val_loss: 0.4518
Epoch 12/25
```