

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

[14]: vgg16_model.summary()

| Layer (type) | Output shape | Param # |
|----------------------------|-----------------------|-----------|
| input_layer_1 (InputLayer) | (None, 224, 224, 3) | 0 |
| block1_conv1 (Conv2D) | (None, 224, 224, 64) | 1,792 |
| block1_conv2 (Conv2D) | (None, 224, 224, 64) | 86,032 |
| block1_pool (MaxPooling2D) | (None, 112, 112, 64) | 0 |
| block2_conv1 (Conv2D) | (None, 112, 112, 128) | 73,808 |
| block2_conv2 (Conv2D) | (None, 112, 112, 128) | 147,344 |
| block2_pool (MaxPooling2D) | (None, 56, 56, 128) | 0 |
| block3_conv1 (Conv2D) | (None, 56, 56, 256) | 295,360 |
| block3_conv2 (Conv2D) | (None, 56, 56, 256) | 596,800 |
| block3_pool (MaxPooling2D) | (None, 28, 28, 256) | 0 |
| block4_conv1 (Conv2D) | (None, 28, 28, 512) | 1,180,800 |
| block4_conv2 (Conv2D) | (None, 28, 28, 512) | 1,180,800 |
| block4_pool (MaxPooling2D) | (None, 14, 14, 512) | 0 |
| block5_conv1 (Conv2D) | (None, 14, 14, 512) | 1,180,800 |
| block5_conv2 (Conv2D) | (None, 14, 14, 512) | 1,180,800 |
| block5_pool (MaxPooling2D) | (None, 7, 7, 512) | 0 |
| flatten_1 (Flatten) | (None, 25088) | 0 |
| dense_1 (Dense) | (None, 41) | 104,352 |

Total params: 14,733,184 (36.31 MB)
Trainable params: 108,156 (262.40 KB)
Non-trainable params: 14,734,028 (36.33 MB)

[15]: model=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: 0.0784 - val_accuracy: 0.9171 - val_loss: 0.0244 |
| Epoch 2/15 | 144/144 | 5187s 36s/step - accuracy: 0.8084 - loss: 0.0324 - val_accuracy: 0.9283 - val_loss: 0.0229 |
| Epoch 3/15 | 144/144 | 399s 3s/step - accuracy: 0.9364 - loss: 0.0217 - val_accuracy: 0.9328 - 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: 0.9498 - val_loss: 0.0298 |
| Epoch 8/15 | 144/144 | 303s 2s/step - accuracy: 0.9517 - loss: 0.0268 - val_accuracy: 0.9479 - val_loss: 0.0242 |
| Epoch 9/15 | 144/144 | 286s 2s/step - accuracy: 0.9503 - loss: 0.0208 - val_accuracy: 0.9486 - val_loss: 0.0253 |

[16]: keras.src.callbacks.history.History at 0x2409f1369f90

Resnet-50

[17]: resnet50_model.summary()

| Layer (type) | Output shape | Param # | Connected to |
|--|----------------------|---------|---------------------------|
| input_layer (InputLayer) | (None, 224, 224, 3) | 0 | - |
| conv1_pad (ConvPadding2D) | (None, 224, 224, 3) | 0 | input_layer[0][0] |
| conv1_conv (Conv2D) | (None, 112, 112, 64) | 8,472 | conv1_pad[0][0] |
| conv1_bn (BatchNormalisation) | (None, 112, 112, 64) | 256 | conv1_conv[0][0] |
| conv1_relu (Activation) | (None, 112, 112, 64) | 0 | conv1_bn[0][0] |
| pool1_pad (ConvPadding2D) | (None, 112, 112, 64) | 0 | conv1_relu[0][0] |
| pool1_pool (MaxPooling2D) | (None, 56, 56, 64) | 0 | pool1_pad[0][0] |
| conv2_block1_1_conv (Conv2D) | (None, 56, 56, 64) | 4,480 | pool1_pool[0][0] |
| conv2_block1_1_bn (BatchNormalisation) | (None, 56, 56, 64) | 256 | conv2_block1_1_conv[0][0] |
| conv2_block1_1_relu (Activation) | (None, 56, 56, 64) | 0 | conv2_block1_1_bn[0][0] |
| conv2_block1_2_conv (Conv2D) | (None, 56, 56, 64) | 16,400 | conv2_block1_1_relu[0][0] |
| conv2_block1_2_bn (BatchNormalisation) | (None, 56, 56, 64) | 256 | conv2_block1_2_conv[0][0] |
| conv2_block1_2_relu (Activation) | (None, 56, 56, 64) | 0 | conv2_block1_2_bn[0][0] |
| conv2_block1_3_conv (Conv2D) | (None, 56, 56, 256) | 16,400 | pool1_pool[0][0] |
| conv2_block1_3_bn (BatchNormalisation) | (None, 56, 56, 256) | 1,600 | conv2_block1_3_conv[0][0] |
| conv2_block1_3_relu (Activation) | (None, 56, 56, 256) | 0 | conv2_block1_3_bn[0][0] |
| conv2_block1_3_bn (BatchNormalisation) | (None, 56, 56, 256) | 1,600 | conv2_block1_3_conv[0][0] |

[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])

| | | |
|-------------|---------|--|
| Epoch 1/20 | 144/144 | 344s 2s/step - accuracy: 0.8804 - loss: 0.0206 - val_accuracy: 0.9514 - val_loss: 0.0055 |
| Epoch 2/20 | 144/144 | 331s 2s/step - accuracy: 0.9489 - loss: 0.0086 - val_accuracy: 0.9307 - val_loss: 0.0147 |
| Epoch 3/20 | 144/144 | 337s 2s/step - accuracy: 0.9531 - loss: 0.0108 - val_accuracy: 0.9421 - val_loss: 0.0118 |
| Epoch 4/20 | 144/144 | 336s 2s/step - accuracy: 0.9557 - loss: 0.0100 - val_accuracy: 0.9521 - val_loss: 0.0105 |
| Epoch 5/20 | 144/144 | 709s 5s/step - accuracy: 0.9558 - loss: 0.0108 - val_accuracy: 0.9486 - val_loss: 0.0115 |
| Epoch 6/20 | 144/144 | 367s 3s/step - accuracy: 0.9636 - loss: 0.0084 - val_accuracy: 0.9243 - val_loss: 0.0277 |
| Epoch 7/20 | 144/144 | 340s 2s/step - accuracy: 0.9658 - loss: 0.0101 - val_accuracy: 0.9579 - val_loss: 0.0109 |
| Epoch 8/20 | 144/144 | 330s 2s/step - accuracy: 0.9649 - loss: 0.0117 - val_accuracy: 0.9787 - val_loss: 0.0110 |
| Epoch 9/20 | 144/144 | 330s 2s/step - accuracy: 0.9717 - loss: 0.0094 - val_accuracy: 0.9658 - val_loss: 0.0114 |
| Epoch 10/20 | 144/144 | 330s 2s/step - accuracy: 0.9718 - loss: 0.0110 - val_accuracy: 0.9721 - val_loss: 0.0114 |
| Epoch 11/20 | 144/144 | 331s 2s/step - accuracy: 0.9695 - loss: 0.0115 - val_accuracy: 0.9743 - val_loss: 0.0089 |
| Epoch 12/20 | 144/144 | 365s 3s/step - accuracy: 0.9756 - loss: 0.0089 - val_accuracy: 0.9688 - val_loss: 0.0135 |
| Epoch 13/20 | 144/144 | 341s 2s/step - accuracy: 0.9725 - loss: 0.0099 - val_accuracy: 0.9571 - val_loss: 0.0201 |
| Epoch 14/20 | 144/144 | 378s 3s/step - accuracy: 0.9698 - loss: 0.0127 - val_accuracy: 0.9686 - val_loss: 0.0158 |

[22]: keras.src.callbacks.history.History at 0x259081c1f590

| resnet18_model.summary() | | | | |
|---|--------------------|-----------|------------------------|--|
| conv1_block1_relu | (None, 7, 7, 512) | 0 | conv1_block1_1_act[0] | |
| conv1_block1_conv (conv2d) | (None, 7, 7, 512) | 2,159,390 | conv1_block1_1_relu[0] | |
| conv1_block1_bn (BatchNormalization) | (None, 7, 7, 512) | 2,080 | conv1_block1_1_act[0] | |
| conv1_block1_relu (activation) | (None, 7, 7, 512) | 0 | conv1_block1_1_bn[0] | |
| conv1_block1_conv (conv2d) | (None, 7, 7, 2048) | 1,656,624 | conv1_block1_1_relu[0] | |
| conv1_block1_bn (BatchNormalization) | (None, 7, 7, 2048) | 8,192 | conv1_block1_1_act[0] | |
| conv1_block1_add (add) | (None, 7, 7, 2048) | 0 | conv1_block1_1_bn[0] | |
| conv1_block1_act (activation) | (None, 7, 7, 2048) | 0 | conv1_block1_1_act[0] | |
| conv1_block1_conv (conv2d) | (None, 7, 7, 512) | 1,945,080 | conv1_block1_1_act[0] | |
| conv1_block1_bn (BatchNormalization) | (None, 7, 7, 512) | 2,080 | conv1_block1_1_act[0] | |
| conv1_block1_relu (activation) | (None, 7, 7, 512) | 0 | conv1_block1_1_bn[0] | |
| conv1_block1_conv (conv2d) | (None, 7, 7, 512) | 2,159,390 | conv1_block1_1_relu[0] | |
| conv1_block1_bn (BatchNormalization) | (None, 7, 7, 512) | 2,080 | conv1_block1_1_act[0] | |
| conv1_block1_relu (activation) | (None, 7, 7, 512) | 0 | conv1_block1_1_bn[0] | |
| conv1_block1_conv (conv2d) | (None, 7, 7, 2048) | 1,656,624 | conv1_block1_1_relu[0] | |
| conv1_block1_bn (BatchNormalization) | (None, 7, 7, 2048) | 8,192 | conv1_block1_1_act[0] | |
| conv1_block1_add (add) | (None, 7, 7, 2048) | 0 | conv1_block1_1_bn[0] | |
| conv1_block1_act (activation) | (None, 7, 7, 2048) | 0 | conv1_block1_1_act[0] | |
| flatten (flatten) | (None, 5, 5, 3840) | 0 | conv1_block1_1_bn[0] | |
| flatten (flatten) | (None, 3840) | 0 | conv1_block1_1_act[0] | |
| dense (dense) | (None, 4) | 403,412 | flatten[0] | |
| total params: 21,509,124 (50.51 MB) | | | | |
| trainable params: 404,412 (1.53 MB) | | | | |
| non-trainable params: 21,104,712 (50.98 MB) | | | | |

[illegible]

Efficient Net

| dflc[latency_model_summary] | | | | | |
|---|--------------------|---------|----------------------------|--|--|
| latency_model_summary | | | | | |
| isoclock_prop (propval) | (None, 5, 7, 1962) | 0 | isoclock_propact[0][0] | | |
| isoclock_amb (amb) | (None, 5, 7, 1962) | 0 | isoclock_propact[0][1] | | |
| isoclock_expand_cmc (cmc) | (None, 5, 7, 1332) | 221.384 | isoclock_expand_cm[0] | | |
| isoclock_expand_cm (batchnormalization) | (None, 5, 7, 1332) | 4.688 | isoclock_expand_cm[0][0] | | |
| isoclock_expand_activation | (None, 5, 7, 1332) | 0 | isoclock_expand_cm[0][1] | | |
| isoclock_ambcm (batchnormalization) | (None, 5, 7, 1332) | 16.368 | isoclock_expand_activation | | |
| isoclock_amb (batchnormalization) | (None, 5, 7, 1332) | 4.688 | isoclock_expand_cm[0][0] | | |
| isoclock_activation (activation) | (None, 5, 7, 1332) | 0 | isoclock_expand_cm[0][1] | | |
| isoclock_expand_activation | (None, 1332) | 0 | isoclock_activation[0][0] | | |
| isoclock_expand_activation (batchnormalization) | (None, 5, 7, 1332) | 0 | isoclock_expand_cm[0][0] | | |
| isoclock_expand_cm (cmc) | (None, 5, 7, 1332) | 16.364 | isoclock_expand_cm[0][0] | | |
| isoclock_expand_cm (batchnormalization) | (None, 5, 7, 1332) | 16.448 | isoclock_expand_cm[0][0] | | |
| isoclock_expand_activation (batchnormalization) | (None, 5, 7, 1332) | 0 | isoclock_expand_cm[0][0] | | |
| isoclock_expand_cm (cmc) | (None, 5, 7, 1332) | 16.448 | isoclock_expand_cm[0][0] | | |
| isoclock_expand_cm (batchnormalization) | (None, 5, 7, 1332) | 16.448 | isoclock_expand_cm[0][0] | | |
| top_cm (cmc) | (None, 5, 7, 1280) | 400.000 | isoclock_expand_cm[0][0] | | |
| top_cm (batchnormalization) | (None, 5, 7, 1280) | 5.128 | top_cm[0][0] | | |
| top_activation (activation) | (None, 5, 7, 1280) | 0 | top_cm[0][0] | | |
| flattten (flattten) | (None, 62768) | 0 | top_activation[0][0] | | |
| dense (dense) | (None, 4) | 208.584 | flattten[0][0] | | |
| total params: 4,902,325 (10.32 MB) | | | | | |
| inference params: 256,416 (0.504 MB) | | | | | |
| non-trainable params: 4,645,909 (10.45 MB) | | | | | |
| optimizer params: 685,376 (1.35 MB) | | | | | |