# Task 4

## a)

To resize all images to 224x224x3, we can use the Python Imaging Library (PIL) module. The following code can be used to resize all the images in the dataset:

CSS CODE:  
from PIL import Image

import os

data\_dir = '/path/to/dataset'

for img\_name in os.listdir(data\_dir):

img\_path = os.path.join(data\_dir, img\_name)

img = Image.open(img\_path)

img = img.resize((224, 224))

img.save(img\_path)

## b)

To train the base model architecture from Task 1 with the skin cancer dataset, we can use the same code as in Task 1, with the modifications to the input shape and the number of output classes:

import tensorflow as tf

from tensorflow.keras import layers, models

# Define the base model architecture

model = models.Sequential()

model.add(layers.Conv2D(32, (3, 3), activation='relu', input\_shape=(224, 224, 3)))

model.add(layers.MaxPooling2D((2, 2)))

model.add(layers.Conv2D(64, (3, 3), activation='relu'))

model.add(layers.MaxPooling2D((2, 2)))

model.add(layers.Conv2D(128, (3, 3), activation='relu'))

model.add(layers.MaxPooling2D((2, 2)))

model.add(layers.Conv2D(128, (3, 3), activation='relu'))

model.add(layers.MaxPooling2D((2, 2)))

model.add(layers.Flatten())

model.add(layers.Dense(512, activation='relu'))

model.add(layers.Dense(2, activation='softmax'))

# Compile the model

model.compile(optimizer='adam',

loss='categorical\_crossentropy',

metrics=['accuracy'])

# Train the model with the skin cancer dataset

train\_datagen = tf.keras.preprocessing.image.ImageDataGenerator(

rescale=1./255,

validation\_split=0.2)

train\_generator = train\_datagen.flow\_from\_directory(

'/path/to/dataset',

target\_size=(224, 224),

batch\_size=32,

class\_mode='categorical',

subset='training')

validation\_generator = train\_datagen.flow\_from\_directory(

'/path/to/dataset',

target\_size=(224, 224),

batch\_size=32,

class\_mode='categorical',

subset='validation')

model.fit(

train\_generator,

steps\_per\_epoch=train\_generator.samples//train\_generator.batch\_size,

validation\_data=validation\_generator,

validation\_steps=validation\_generator.samples//validation\_generator.batch\_size,

epochs=10)

# Evaluate the model with the testing dataset

test\_generator = train\_datagen.flow\_from\_directory(

'/path/to/testing/dataset',

target\_size=(224, 224),

batch\_size=32,

class\_mode='categorical')

loss, accuracy = model.evaluate(test\_generator)

print(f'Testing accuracy: {accuracy\*100:.2f}%')

To compute sensitivity and specificity, we need to calculate the number of true positives (TP), false positives (FP), true negatives (TN), and false negatives (FN) from the confusion matrix. The sensitivity is the ratio of TP to TP+FN, and the specificity is the ratio of TN to TN+FP.

import numpy as np

from sklearn.metrics import confusion\_matrix, roc\_curve, auc

# Predict the classes of the testing dataset

y\_pred = model.predict(test\_generator)

y\_true = test\_generator.classes

# Compute the confusion matrix

cm = confusion\_matrix(y\_true, np.argmax(y\_pred, axis=1))

# Compute sensitivity and specificity

TP = cm[1, 1]

FN = cm[1, 0]

TN = cm[0, 0]

FP = cm[0, 1]

sensitivity = TP / (TP + FN)

specificity = TN / (TN + FP)

print(f'Sensitivity: {sensitivity\*100:.2f}%')

print(f'Specificity: {specificity\*100:.2f}%')

This code assumes that the predicted classes from the model are in the form of one-hot encoded vectors, so we need to convert them back to class labels using np.argmax(y\_pred, axis=1) before computing the confusion matrix.

## c)

To use the pre-trained VGG16 model as a feature extractor, we can remove the last layer of the VGG16 model and use the output of the second last layer as features. Here is the code to compute the features of all the images in the dataset:

css

Copy code

from tensorflow.keras.applications.vgg16 import VGG16, preprocess\_input

from tensorflow.keras.preprocessing import image

import numpy as np

import os

data\_dir = '/path/to/dataset'

# Load the pre-trained VGG16 model

base\_model = VGG16(weights='imagenet', include\_top=False)

# Extract features from the images in the dataset

features = []

labels = []

for img\_name in os.listdir(data\_dir):

img\_path = os.path.join(data\_dir, img\_name)

img = image.load\_img(img\_path, target\_size=(224, 224))

x = image.img\_to\_array(img)

x = np.expand\_dims(x, axis=0)

x = preprocess\_input(x)

features.append(base\_model.predict(x).flatten())

labels.append(int('cancer' in img\_name))

features = np.array(features)

labels = np.array(labels)

## d)

To train an SVM classifier with the features computed in part c, we can use the scikit-learn library. Here is the code to train and evaluate the SVM classifier:

makefile

Copy code

from sklearn.svm import SVC

from sklearn.metrics import confusion\_matrix, roc\_curve, auc

from sklearn.model\_selection import train\_test\_split

# Split the dataset into training and testing sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(features, labels, test\_size=0.2, random\_state=42)

# Train the SVM classifier

svm = SVC(kernel='linear', probability=True)

svm.fit(X\_train, y\_train)

# Evaluate the SVM classifier

y\_pred = svm.predict(X\_test)

y\_prob = svm.predict\_proba(X\_test)[:, 1]

# Compute the confusion matrix

cm = confusion\_matrix(y\_test, y\_pred)

# Compute sensitivity and specificity

TP = cm[1, 1]

FN = cm[1, 0]

TN = cm[0, 0]

FP = cm[0, 1]

sensitivity = TP / (TP + FN)

specificity = TN / (TN + FP)

# Compute the ROC curve and AUC

fpr, tpr, thresholds = roc\_curve(y\_test, y\_prob)

roc\_auc = auc(fpr, tpr)

## e)

To fine-tune the pre-trained VGG16 model with the skin cancer dataset, we can use the same code as in Task 3, with the modifications to the input shape and the number of output classes:

python

Copy code

import tensorflow as tf

from tensorflow.keras import layers, models

from tensorflow.keras.applications.vgg16 import VGG16

# Load the pre-trained VGG16 model

base\_model = VGG16(weights='imagenet', include\_top=False, input\_shape=(224, 224, 3))

# Add new layers for fine-tuning

model = models.Sequential()

model.add(base\_model)

model.add(layers.Flatten())

model.add(layers.Dense(256, activation='relu'))

model.add(layers.Dropout(0.5))

model.add(layers.Dense(1, activation='sigmoid'))

# Freeze the layers of the pre-trained model

base\_model.trainable = False

# Compile the model

model.compile(optimizer=tf.keras.optimizers.Adam(lr=1e-5),

loss='binary\_crossentropy',

metrics=['accuracy'])

# Train the model