# IMPORTANT: RUN THIS CELL IN ORDER TO IMPORT YOUR KAGGLE DATA SOURCES,

# THEN FEEL FREE TO DELETE THIS CELL.

# NOTE: THIS NOTEBOOK ENVIRONMENT DIFFERS FROM KAGGLE'S PYTHON

# ENVIRONMENT SO THERE MAY BE MISSING LIBRARIES USED BY YOUR

# NOTEBOOK.

import kagglehub

shivam17299\_oral\_cancer\_lips\_and\_tongue\_images\_path = kagglehub.dataset\_download('shivam17299/oral-cancer-lips-and-tongue-images')

print('Data source import complete.')

from google.colab import drive

drive.mount('/content/drive')

pip install -U tensorflow-addons

!pip install visualkeras

import os

import warnings

import itertools

import cv2

import seaborn as sns

import pandas as pd

import numpy as np

from PIL import Image

from sklearn.utils import class\_weight

from sklearn.metrics import confusion\_matrix, classification\_report

from collections import Counter

import tensorflow as tf

import visualkeras

import plotly.express as px

import matplotlib.pyplot as plt

from sklearn.metrics import multilabel\_confusion\_matrix

from tensorflow.keras.preprocessing.image import load\_img

from tensorflow.keras.utils import plot\_model

from tensorflow.keras import layers

from tensorflow.keras import regularizers

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

warnings.filterwarnings('ignore')

%matplotlib inline

## Setting up general parameters

# General parameters

epochs = 20

image\_size = 240

np.random.seed(42)

tf.random.set\_seed(42)

## Data Loading, Preperation and Visualization

os.listdir("/content/drive/MyDrive/OralCancer minor",)

folder\_path = ("/content/drive/MyDrive/OralCancer minor")

import os

import cv2

import numpy as np

def load\_and\_process\_dataset(folder\_path):

"""Loads and processes images using alternative while loop structures."""

dataset = []

labels = []

class\_folders = ['non-cancer', 'cancer']

class\_index = 0

while class\_index < len(class\_folders):

class\_folder = class\_folders[class\_index]

images\_path = os.path.join(folder\_path, class\_folder)

image\_index = 0

while True: # Loop infinitely until a "break" occurs

try:

image\_name = os.listdir(images\_path)[image\_index] # Access by index

image\_path = os.path.join(images\_path, image\_name)

image = cv2.imread(image\_path)

image = cv2.cvtColor(image, cv2.COLOR\_BGR2RGB) # Convert to RGB

image = cv2.resize(image, (240, 240))

dataset.append(image)

labels.append(class\_index) # Use class\_index directly for labels

image\_index += 1

except IndexError: # Handle end of image list

break # Exit the inner loop

class\_index += 1

return np.array(dataset), np.array(labels)

# Assuming `folder\_path` is already a string

dataset, labels = load\_and\_process\_dataset(folder\_path)

# Convert to NumPy arrays

dataset = np.array(dataset)

lab = np.array(labels)

# Print shapes

print(dataset.shape, labels.shape)

# Convert to NumPy arrays and print shapes in a single line

print(np.array(dataset).shape, np.array(labels).shape)

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

x\_train, x\_test, y\_train, y\_test = train\_test\_split(np.array(dataset), np.array(labels), test\_size=0.2, shuffle=True, random\_state=42)

import matplotlib.pyplot as plt

def plot\_state(state):

images = [load\_img(os.path.join(folder\_path, state, img\_name), target\_size=(image\_size, image\_size))

for img\_name in os.listdir(os.path.join(folder\_path, state))[:9]]

fig, axes = plt.subplots(3, 3, figsize=(12, 12))

for ax, img in zip(axes.flat, images):

ax.imshow(img)

plt.show()

plot\_state('cancer')

plot\_state("non-cancer")

# \*\*\*\*\*\* Baseline Model CNN\*\*

from tensorflow.keras.layers import Input, Conv2D, MaxPooling2D, Flatten, Dense, Dropout

from tensorflow.keras import regularizers

from tensorflow.keras.models import Model

# Input layer

input\_layer = Input(shape=(image\_size, image\_size, 3))

# Convolutional layers

conv1 = Conv2D(filters=32, kernel\_size=(3, 3), strides=(2, 2), activation="relu", padding="valid")(input\_layer)

maxpool1 = MaxPooling2D((2, 2))(conv1)

conv2 = Conv2D(filters=32, kernel\_size=(3, 3), strides=(2, 2), activation="relu", padding="valid")(maxpool1)

maxpool2 = MaxPooling2D((2, 2))(conv2)

# Flatten layer

flatten = Flatten()(maxpool2)

# Dense layers with regularization

dense1 = Dense(units=64, activation='relu',

kernel\_regularizer=regularizers.L1L2(l1=1e-3, l2=1e-3),

bias\_regularizer=regularizers.L2(1e-2),

activity\_regularizer=regularizers.L2(1e-3))(flatten)

dropout = Dropout(0.5)(dense1)

output\_layer = Dense(units=1, activation='sigmoid')(dropout)

# Define the model

model = Model(inputs=input\_layer, outputs=output\_layer)

from tensorflow.keras.optimizers import Adam

from tensorflow.keras.losses import BinaryCrossentropy

# Compile the model

model.compile(

optimizer=Adam(),

loss=BinaryCrossentropy(),

metrics=['accuracy']

)

# Print the model summary

model.summary()

\*\*VGG16\*\*

# Importing the pre-trained VGG16 model

from tensorflow.keras.applications import VGG16

from tensorflow.keras.layers import Input, Flatten, Dense, Dropout

from tensorflow.keras.models import Model

# Input shape remains the same

input\_layer = Input(shape=(image\_size, image\_size, 3))

# Load the VGG16 model with pre-trained weights, excluding the top classifier layers

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

# Freeze the layers of VGG16 so that their weights won't be updated during training

for layer in base\_model.layers:

layer.trainable = False

# Add custom layers on top of the VGG16 model

x = base\_model.output

x = Flatten()(x) # Flatten the output

x = Dense(64, activation='relu')(x) # Fully connected layer

x = Dropout(0.5)(x) # Dropout layer for regularization

output\_layer = Dense(1, activation='sigmoid')(x) # Output layer for binary classification

# Create the final model

model = Model(inputs=base\_model.input, outputs=output\_layer)

# Compile the model

model.compile(optimizer='adam', loss='binary\_crossentropy', metrics=['accuracy'])

# Print the model summary

model.summary()

plot\_model(model, show\_shapes=True, show\_layer\_names=False)

visualkeras.layered\_view(model, legend=True)

class\_weights = {}

for cls in np.unique(y\_train):

count = np.sum(y\_train == cls)

class\_weights[cls] = 1.0 / count

import tensorflow as tf

from sklearn.utils.class\_weight import compute\_class\_weight

# Assuming 'model', 'x\_train', 'y\_train', 'x\_test', and 'y\_test' are already defined

# Calculate class weights

class\_weights = compute\_class\_weight('balanced', classes=np.unique(y\_train), y=y\_train)

class\_weights\_dict = {0: class\_weights[0], 1: class\_weights[1]}

# Compile the model

model.compile(

optimizer=Adam(),

loss=BinaryCrossentropy(),

metrics=['accuracy']

)

# Train the model

history = model.fit(

x\_train, y\_train,

epochs=200,

class\_weight=class\_weights\_dict,

validation\_data=(x\_test, y\_test),

verbose=1

)

# ## Evaluating CNN Model

import seaborn as sns

sns.set\_style("whitegrid")

fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(12, 5))

ax1.plot(history.history['loss'], label='Training Loss')

ax1.plot(history.history['val\_loss'], label='Validation Loss')

ax1.set\_title('Loss', fontsize=14)

ax1.legend()

ax2.plot(history.history['accuracy'], label='Training Accuracy')

ax2.plot(history.history['val\_accuracy'], label='Validation Accuracy')

ax2.set\_title('Accuracy', fontsize=14)

ax2.legend()

fig.suptitle('Optimizer: Adam', fontsize=16)

plt.show()

# Assuming 'model', 'x\_test', and 'y\_test' are already defined

# Evaluate the model on the test set

test\_loss, test\_accuracy = model.evaluate(x\_test, y\_test)

# Print the accuracy

print('The accuracy of the baseline model CNN is {:.2f}%!'.format(test\_accuracy \* 100))

predictions = model.predict(x\_test)

y\_pred = (predictions >= 0.5).astype(int)

import matplotlib.pyplot as plt

import seaborn as sns

import pandas as pd

def plot\_confusion\_matrix\_enhanced(cm, classes, title='Confusion Matrix', cmap='viridis'):

"""

Plots a confusion matrix with a distinct style and colormap.

"""

# Normalize for better visualization

cm\_norm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]

# Create a DataFrame for flexible plotting

df\_cm = pd.DataFrame(cm\_norm, index=classes, columns=classes)

# Employ Seaborn for a visually appealing heatmap

sns.heatmap(df\_cm, annot=True, fmt='.2f', cmap=cmap, vmin=0, vmax=1)

# Customize labels and title

plt.xlabel('Predicted Label')

plt.ylabel('True Label')

plt.title(title)

plt.show()

import matplotlib.pyplot as plt

import seaborn as sns

import numpy as np

# Compute confusion matrix

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

# Create a hexagonal mask for a visually striking shape

mask = np.triu(np.ones\_like(cnf\_matrix, dtype=bool)) & np.tril(np.ones\_like(cnf\_matrix, dtype=bool), k=-1)

# Plot using Seaborn with distinct customizations

sns.heatmap(

cnf\_matrix,

annot=True,

fmt='.2f',

cmap='cubehelix', # Employ a vibrant colormap

mask=mask,

linewidths=0.5,

cbar=False,

xticklabels=["non-cancer", "cancer"],

yticklabels=["non-cancer", "cancer"]

)

plt.title('Confusion Matrix - Hexagonal Emphasis')

plt.xlabel('Predicted Label')

plt.ylabel('True Label')

plt.show()