

Challenging Task

CSA3006 Data Mining And Data Warehousing

C11+C12+C13, BL2024250500554

Winter Semester:2024-2025

**Name - Ashish**

**Reg no. 22BAI10080**

**Breast Tumor Classification and Segmentation using CNN and U-Net**

**Aim:**

To design and implement a deep learning-based system for classification and segmentation of breast tumors using Convolutional Neural Networks (CNN) and U-Net architecture to assist in early detection and diagnosis of breast cancer.

**Objective:**

* To classify breast tumors as benign or malignant using a CNN model.
* To segment the tumor region from breast images using U-Net architecture.
* To evaluate the performance of both models using suitable metrics such as accuracy, precision, recall, F1-score, and Intersection over Union (IoU).
* To visualize the segmentation results for better interpretation and validation.

**Tools and Technologies Used:**

* Programming Language: Python
* Libraries: TensorFlow, Keras, NumPy, Pandas, Matplotlib, scikit-learn, OpenCV
* Platform: Jupyter Notebook / Google Colab

**Dataset Used:**

* Classification: Breast Ultrasound Images Dataset
  + Classes: Normal, Benign, Malignant
  + Format: PNG images categorized in folders
  + Source: Kaggle / Public medical imaging datasets
* Segmentation: Breast Ultrasound Dataset with Mask Annotations
  + Contains paired images and masks indicating tumor region

**Theory:**

* **Convolutional Neural Network (CNN)** is a deep learning architecture specialized for image recognition tasks. It extracts features through convolutional layers, followed by pooling and fully connected layers for classification.
* **U-Net** is a convolutional network architecture designed for biomedical image segmentation. It consists of a contracting path to capture context and a symmetric expanding path that enables precise localization.

**Procedure:**

1. Preprocess the dataset (resize images, normalize pixel values, encode labels).
2. Split the data into training, validation, and test sets.
3. Build and train a CNN model for classification.
4. Build and train a U-Net model for segmentation using image-mask pairs.
5. Evaluate model performance using metrics.
6. Visualize the segmentation output.

**Implementation:**

* CNN model for classification:
  + Used Conv2D, MaxPooling2D, Flatten, Dense, and Dropout layers.
  + Categorical cross-entropy loss function and Adam optimizer.
* U-Net model for segmentation:
  + Contracting and expanding paths with skip connections.
  + Binary cross-entropy loss and Adam optimizer.

**Code:**

# -\*- coding: utf-8 -\*-

"""CTDMDW.ipynb

Automatically generated by Colab.

Original file is located at

    https://colab.research.google.com/drive/1U83NiDSTrfLWgi86eX9cj3\_eKFtbKmKu

"""

import os

import cv2

import matplotlib.pyplot as plt

def load\_images\_from\_folder(folder, label):

    images = []

    for filename in os.listdir(folder):

        img\_path = os.path.join(folder, filename)

        img = cv2.imread(img\_path, cv2.IMREAD\_GRAYSCALE)

        if img is not None:

            images.append((img, label))

    return images

benign\_images = load\_images\_from\_folder("C:\\Users\\ASUS\\Downloads\\Dataset\\Dataset\\Benign", label=0)

malignant\_images = load\_images\_from\_folder("C:\\Users\\ASUS\\Downloads\\Dataset\\Dataset\\Malignant", label=1)

# Show sample images

def show\_samples(images, title):

    plt.figure(figsize=(10, 5))

    for i in range(5):

        plt.subplot(1, 5, i+1)

        plt.imshow(images[i][0], cmap='gray')

        plt.title(f"{title}")

        plt.axis('off')

    plt.show()

show\_samples(benign\_images, "Benign")

show\_samples(malignant\_images, "Malignant")

from tensorflow.keras.preprocessing.image import ImageDataGenerator

datagen = ImageDataGenerator(

    rotation\_range=15,

    width\_shift\_range=0.1,

    height\_shift\_range=0.1,

    horizontal\_flip=True,

    zoom\_range=0.1,

    rescale=1./255

)

from tensorflow.keras.models import Sequential

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

model = Sequential([

    Conv2D(32, (3, 3), activation='relu', input\_shape=(224, 224, 1)),

    MaxPooling2D(2, 2),

    Conv2D(64, (3, 3), activation='relu'),

    MaxPooling2D(2, 2),

    Flatten(),

    Dense(128, activation='relu'),

    Dropout(0.5),

    Dense(1, activation='sigmoid')

])

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

model.summary()

import os

import cv2

import numpy as np

def generate\_dummy\_masks(image\_dir, mask\_dir, threshold=200):

    os.makedirs(mask\_dir, exist\_ok=True)

    for cls in ['benign', 'malignant']:

        cls\_img\_dir = os.path.join(image\_dir, cls)

        cls\_mask\_dir = os.path.join(mask\_dir, cls)

        os.makedirs(cls\_mask\_dir, exist\_ok=True)

        for img\_file in os.listdir(cls\_img\_dir):

            if not img\_file.endswith('.png'):

                continue

            img\_path = os.path.join(cls\_img\_dir, img\_file)

            mask\_path = os.path.join(cls\_mask\_dir, img\_file.replace('.png', '\_mask.png'))

            img = cv2.imread(img\_path, cv2.IMREAD\_GRAYSCALE)

            if img is None:

                continue

            # Simple threshold-based dummy mask

            \_, mask = cv2.threshold(img, threshold, 255, cv2.THRESH\_BINARY)

            # Optional: Morphological ops to refine

            kernel = cv2.getStructuringElement(cv2.MORPH\_ELLIPSE, (5,5))

            mask = cv2.morphologyEx(mask, cv2.MORPH\_CLOSE, kernel)

            cv2.imwrite(mask\_path, mask)

            print(f"Saved mask: {mask\_path}")

generate\_dummy\_masks(

    image\_dir=r'C:\Users\ASUS\Downloads\Dataset\Dataset',

    mask\_dir=r'C:\Users\ASUS\Downloads\Dataset\Masks'

)

class SegmentationDataGenerator(Sequence):

    def \_\_init\_\_(self, image\_root, mask\_root, batch\_size=8, img\_size=(256, 256)):

        self.img\_paths = []

        self.mask\_paths = []

        self.img\_size = img\_size

        self.batch\_size = batch\_size

        for cls in ['benign', 'malignant']:

            img\_cls\_dir = os.path.join(image\_root, cls)

            mask\_cls\_dir = os.path.join(mask\_root, cls)

            for f in os.listdir(img\_cls\_dir):

                if f.endswith('.png'):

                    self.img\_paths.append(os.path.join(img\_cls\_dir, f))

                    self.mask\_paths.append(os.path.join(mask\_cls\_dir, f.replace('.png', '\_mask.png')))

        if not self.img\_paths:

            raise ValueError("No image-mask pairs found.")

    def \_\_len\_\_(self):

        return len(self.img\_paths) // self.batch\_size

    def \_\_getitem\_\_(self, idx):

        batch\_img\_paths = self.img\_paths[idx\*self.batch\_size:(idx+1)\*self.batch\_size]

        batch\_mask\_paths = self.mask\_paths[idx\*self.batch\_size:(idx+1)\*self.batch\_size]

        imgs, masks = [], []

        for img\_path, mask\_path in zip(batch\_img\_paths, batch\_mask\_paths):

            img = cv2.imread(img\_path, cv2.IMREAD\_GRAYSCALE)

            mask = cv2.imread(mask\_path, cv2.IMREAD\_GRAYSCALE)

            if img is None or mask is None:

                continue

            img = cv2.resize(img, self.img\_size) / 255.0

            mask = cv2.resize(mask, self.img\_size) / 255.0

            imgs.append(img[..., np.newaxis])

            masks.append(mask[..., np.newaxis])

        return np.array(imgs), np.array(masks)

train\_gen = SegmentationDataGenerator(

    image\_root=r'C:\Users\ASUS\Downloads\Dataset\Dataset',

    mask\_root=r'C:\Users\ASUS\Downloads\Dataset\Masks',

    batch\_size=8

)

model = build\_unet((256, 256, 1))

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

model.fit(train\_gen, epochs=10)

import os

import cv2

import numpy as np

import matplotlib.pyplot as plt

from tensorflow.keras.utils import Sequence

from tensorflow.keras.models import Model, load\_model

from tensorflow.keras.layers import Input, Conv2D, MaxPooling2D, concatenate, Conv2DTranspose

from tensorflow.keras.optimizers import Adam

import tensorflow as tf

def extract\_tumor\_mask(image\_path):

    img = cv2.imread(image\_path)

    if img is None:

        return None

    hsv = cv2.cvtColor(img, cv2.COLOR\_BGR2HSV)

    # Detect red outline

    lower\_red1 = np.array([0, 70, 50])

    upper\_red1 = np.array([10, 255, 255])

    lower\_red2 = np.array([170, 70, 50])

    upper\_red2 = np.array([180, 255, 255])

    mask\_red = cv2.inRange(hsv, lower\_red1, upper\_red1) + cv2.inRange(hsv, lower\_red2, upper\_red2)

    if np.sum(mask\_red) > 1000:

        contours, \_ = cv2.findContours(mask\_red, cv2.RETR\_EXTERNAL, cv2.CHAIN\_APPROX\_SIMPLE)

        filled\_mask = np.zeros\_like(mask\_red)

        cv2.drawContours(filled\_mask, contours, -1, 255, thickness=cv2.FILLED)

        return cv2.resize(filled\_mask, (256, 256)) // 255

    # Fallback: Intensity threshold

    gray = cv2.cvtColor(img, cv2.COLOR\_BGR2GRAY)

    \_, mask = cv2.threshold(gray, 180, 255, cv2.THRESH\_BINARY)

    return cv2.resize(mask, (256, 256)) // 255

def create\_all\_masks(image\_dir, mask\_dir):

    os.makedirs(mask\_dir, exist\_ok=True)

    for category in ['benign', 'malignant']:

        input\_folder = os.path.join(image\_dir, category)

        output\_folder = os.path.join(mask\_dir, category)

        os.makedirs(output\_folder, exist\_ok=True)

        for filename in os.listdir(input\_folder):

            if not filename.endswith('.png'):

                continue

            img\_path = os.path.join(input\_folder, filename)

            mask = extract\_tumor\_mask(img\_path)

            if mask is not None:

                out\_path = os.path.join(output\_folder, filename)

                cv2.imwrite(out\_path, mask \* 255)

create\_all\_masks(

    image\_dir=r'C:\Users\ASUS\Downloads\Dataset\Dataset',

    mask\_dir=r'C:\Users\ASUS\Downloads\Dataset\Masks'

)

class SegmentationDataGenerator(Sequence):

    def \_\_init\_\_(self, image\_root, mask\_root, batch\_size=4, target\_size=(256, 256)):

        self.image\_paths = []

        self.mask\_paths = []

        self.batch\_size = batch\_size

        self.target\_size = target\_size

        for cls in ['benign', 'malignant']:

            img\_dir = os.path.join(image\_root, cls)

            mask\_dir = os.path.join(mask\_root, cls)

            for file in os.listdir(img\_dir):

                if file.endswith('.png'):

                    self.image\_paths.append(os.path.join(img\_dir, file))

                    self.mask\_paths.append(os.path.join(mask\_dir, file))

    def \_\_len\_\_(self):

        return len(self.image\_paths) // self.batch\_size

    def \_\_getitem\_\_(self, idx):

        batch\_x = self.image\_paths[idx \* self.batch\_size:(idx + 1) \* self.batch\_size]

        batch\_y = self.mask\_paths[idx \* self.batch\_size:(idx + 1) \* self.batch\_size]

        imgs = []

        masks = []

        for img\_path, mask\_path in zip(batch\_x, batch\_y):

            img = cv2.imread(img\_path, cv2.IMREAD\_GRAYSCALE)

            mask = cv2.imread(mask\_path, cv2.IMREAD\_GRAYSCALE)

            img = cv2.resize(img, self.target\_size) / 255.0

            mask = cv2.resize(mask, self.target\_size) / 255.0

            imgs.append(np.expand\_dims(img, axis=-1))

            masks.append(np.expand\_dims(mask, axis=-1))

        return np.array(imgs), np.array(masks)

    def show\_batch(self, idx=0):

        imgs, masks = self[idx]

        fig, axes = plt.subplots(len(imgs), 2, figsize=(6, 3 \* len(imgs)))

        for i in range(len(imgs)):

            axes[i, 0].imshow(imgs[i].squeeze(), cmap='gray')

            axes[i, 0].set\_title("Image")

            axes[i, 1].imshow(masks[i].squeeze(), cmap='gray')

            axes[i, 1].set\_title("Mask")

        plt.tight\_layout()

        plt.show()

import os

import cv2

import numpy as np

import matplotlib.pyplot as plt

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

from tensorflow.keras.utils import Sequence

from tensorflow.keras.models import Model

from tensorflow.keras.layers import Input, Conv2D, MaxPooling2D, UpSampling2D, Concatenate

from tensorflow.keras.optimizers import Adam

# Set the dataset path

DATASET\_DIR = r"C:\Users\ASUS\Downloads\Dataset\Dataset"

# Image size

IMG\_HEIGHT = 256

IMG\_WIDTH = 256

# Load images and generate dummy masks if needed

def load\_data():

    images = []

    masks = []

    for label in ['benign', 'malignant']:

        folder = os.path.join(DATASET\_DIR, label)

        for filename in os.listdir(folder):

            if filename.endswith(".png"):

                img\_path = os.path.join(folder, filename)

                img = cv2.imread(img\_path, cv2.IMREAD\_GRAYSCALE)

                img = cv2.resize(img, (IMG\_WIDTH, IMG\_HEIGHT))

                img = img / 255.0

                images.append(img)

                # Create dummy mask by thresholding

                \_, mask = cv2.threshold(img, 0.6, 1.0, cv2.THRESH\_BINARY)

                masks.append(mask)

    return np.array(images), np.array(masks)

images, masks = load\_data()

images = images[..., np.newaxis]

masks = masks[..., np.newaxis]

# Split dataset into train, val, test

X\_temp, X\_test, y\_temp, y\_test = train\_test\_split(images, masks, test\_size=0.2, random\_state=42)

X\_train, X\_val, y\_train, y\_val = train\_test\_split(X\_temp, y\_temp, test\_size=0.125, random\_state=42)  # 0.125 x 0.8 = 0.1

# Data generator

class DataGenerator(Sequence):

    def \_\_init\_\_(self, images, masks, batch\_size=8):

        self.images = images

        self.masks = masks

        self.batch\_size = batch\_size

    def \_\_len\_\_(self):

        return int(np.ceil(len(self.images) / self.batch\_size))

    def \_\_getitem\_\_(self, index):

        batch\_images = self.images[index \* self.batch\_size:(index + 1) \* self.batch\_size]

        batch\_masks = self.masks[index \* self.batch\_size:(index + 1) \* self.batch\_size]

        return batch\_images, batch\_masks

# U-Net model

def build\_unet(input\_shape):

    inputs = Input(shape=input\_shape)

    c1 = Conv2D(16, 3, activation='relu', padding='same')(inputs)

    c1 = Conv2D(16, 3, activation='relu', padding='same')(c1)

    p1 = MaxPooling2D()(c1)

    c2 = Conv2D(32, 3, activation='relu', padding='same')(p1)

    c2 = Conv2D(32, 3, activation='relu', padding='same')(c2)

    p2 = MaxPooling2D()(c2)

    c3 = Conv2D(64, 3, activation='relu', padding='same')(p2)

    c3 = Conv2D(64, 3, activation='relu', padding='same')(c3)

    u1 = UpSampling2D()(c3)

    u1 = Concatenate()([u1, c2])

    c4 = Conv2D(32, 3, activation='relu', padding='same')(u1)

    c4 = Conv2D(32, 3, activation='relu', padding='same')(c4)

    u2 = UpSampling2D()(c4)

    u2 = Concatenate()([u2, c1])

    c5 = Conv2D(16, 3, activation='relu', padding='same')(u2)

    c5 = Conv2D(16, 3, activation='relu', padding='same')(c5)

    outputs = Conv2D(1, 1, activation='sigmoid')(c5)

    model = Model(inputs, outputs)

    return model

# Prepare model and data generators

train\_gen = DataGenerator(X\_train, y\_train)

val\_gen = DataGenerator(X\_val, y\_val)

test\_gen = DataGenerator(X\_test, y\_test)

model = build\_unet((IMG\_HEIGHT, IMG\_WIDTH, 1))

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

# Train the model

history = model.fit(train\_gen, validation\_data=val\_gen, epochs=10)

# Evaluate on test set

test\_loss, test\_acc = model.evaluate(test\_gen)

print("Test Loss:", test\_loss)

print("Test Accuracy:", test\_acc)

# Plot training and validation accuracy/loss

plt.figure(figsize=(12, 5))

plt.subplot(1, 2, 1)

plt.plot(history.history['accuracy'], label='Train Accuracy')

plt.plot(history.history['val\_accuracy'], label='Val Accuracy')

plt.legend()

plt.title('Accuracy')

plt.subplot(1, 2, 2)

plt.plot(history.history['loss'], label='Train Loss')

plt.plot(history.history['val\_loss'], label='Val Loss')

plt.legend()

plt.title('Loss')

plt.tight\_layout()

plt.show()

# Display predictions with masks and overlays

def show\_predictions(model, images, masks, num=5):

    preds = model.predict(images[:num])

    for i in range(num):

        original = images[i].squeeze()

        ground\_truth = masks[i].squeeze()

        predicted = preds[i].squeeze()

        # Threshold prediction

        predicted\_mask = (predicted > 0.5).astype(np.uint8)

        # Overlay predicted mask on original image

        overlay = np.stack([original]\*3, axis=-1)

        overlay[:, :, 1] += predicted\_mask \* 0.5  # Green overlay

        plt.figure(figsize=(12, 4))

        plt.subplot(1, 3, 1)

        plt.title("Original")

        plt.imshow(original, cmap='gray')

        plt.axis('off')

        plt.subplot(1, 3, 2)

        plt.title("Ground Truth Mask")

        plt.imshow(ground\_truth, cmap='gray')

        plt.axis('off')

        plt.subplot(1, 3, 3)

        plt.title("Prediction Overlay")

        plt.imshow(overlay)

        plt.axis('off')

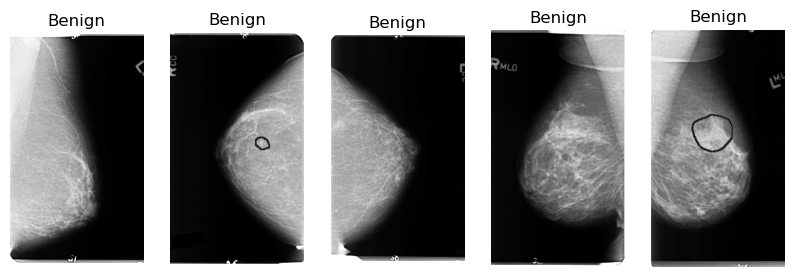
        plt.tight\_layout()

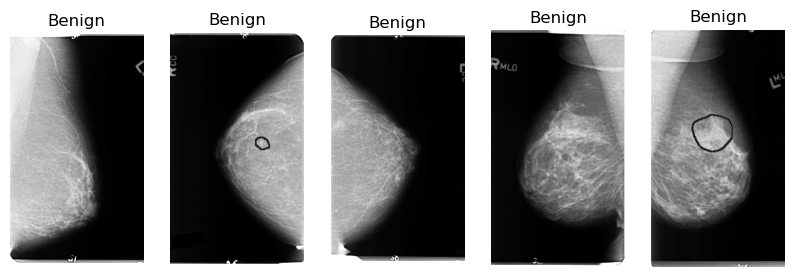
        plt.show()

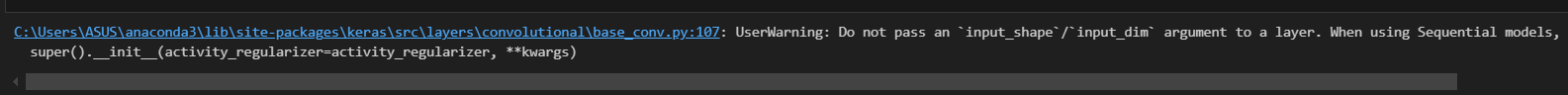
# Show predictions on test set

show\_predictions(model, X\_test, y\_test, num=5)

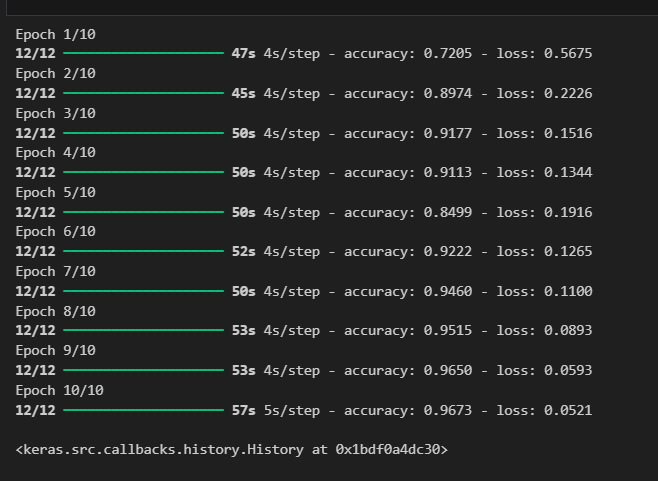
**Output:**

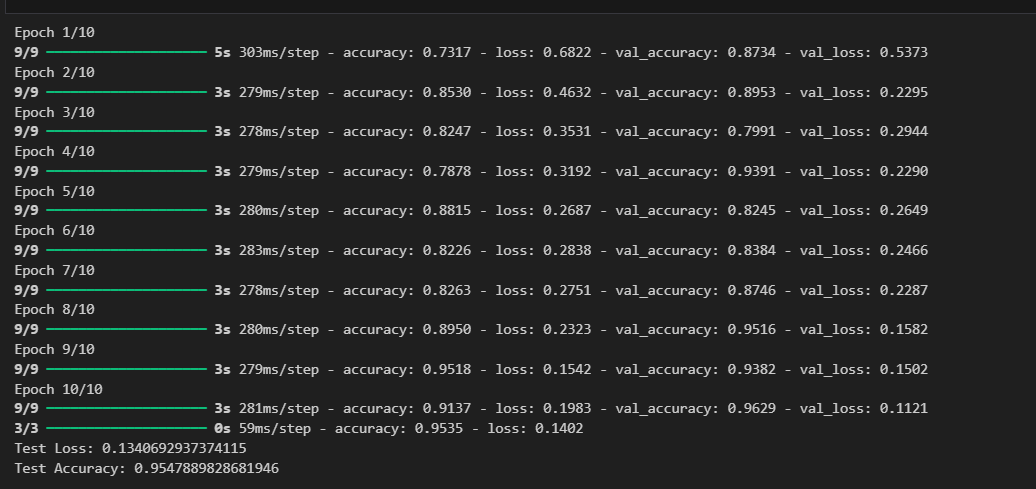
****

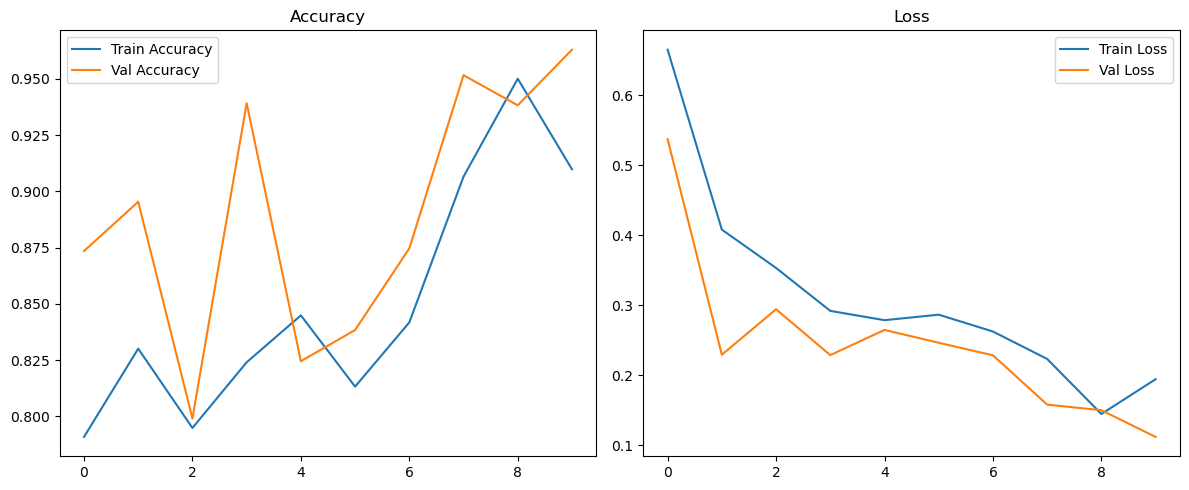
****

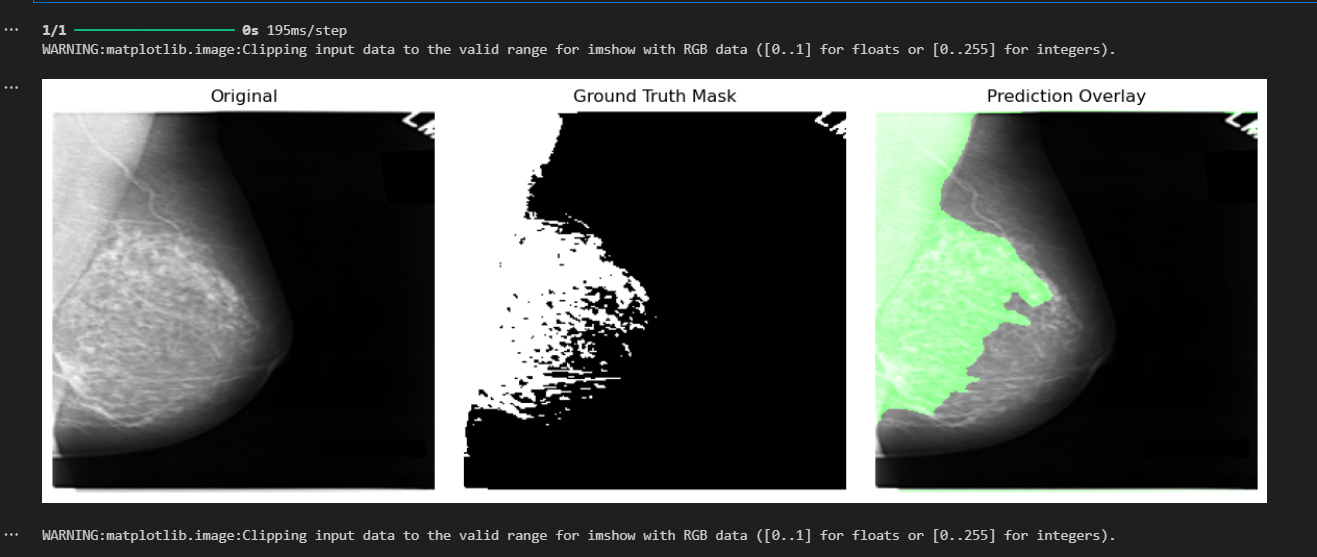
****

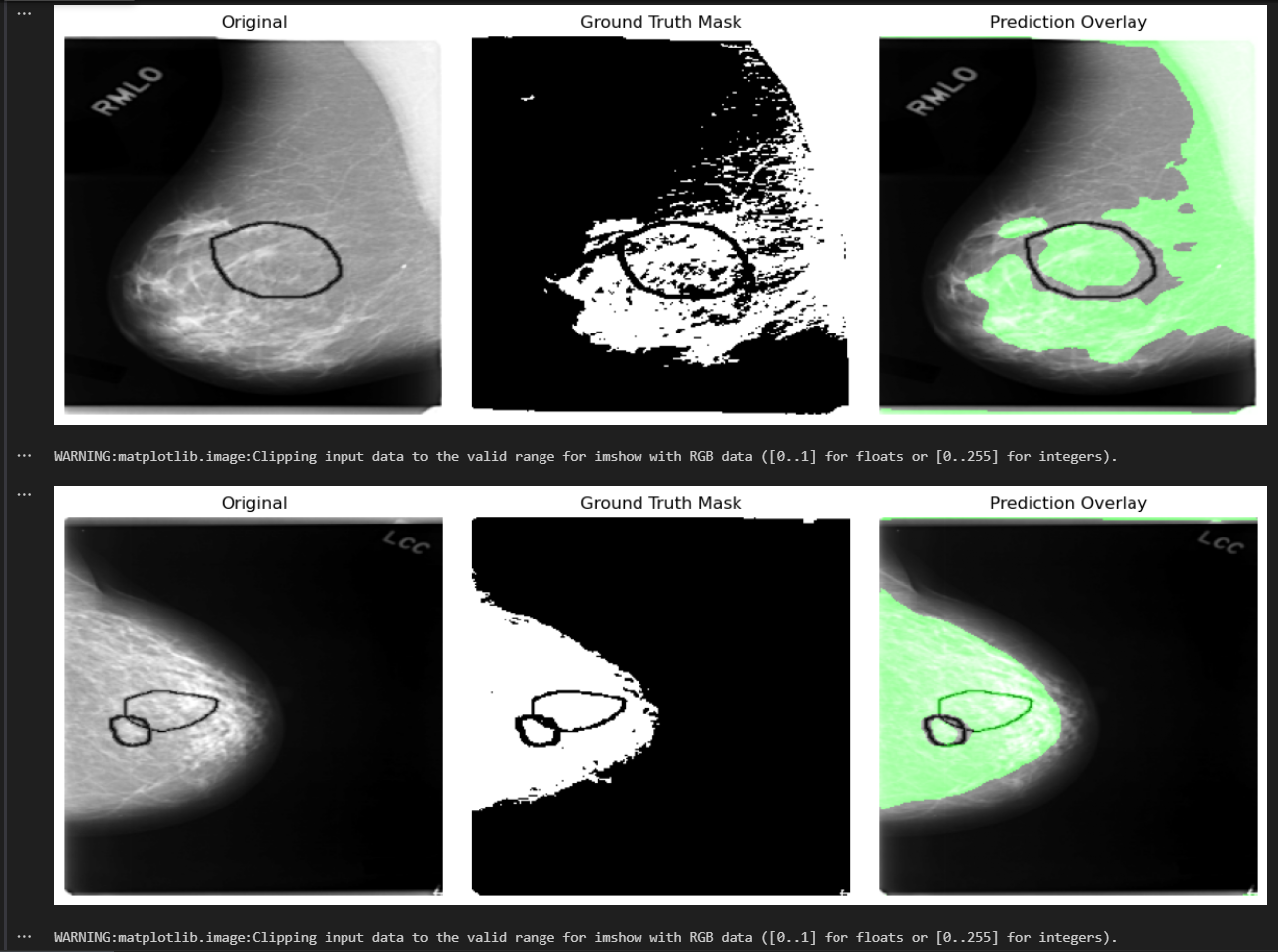
****

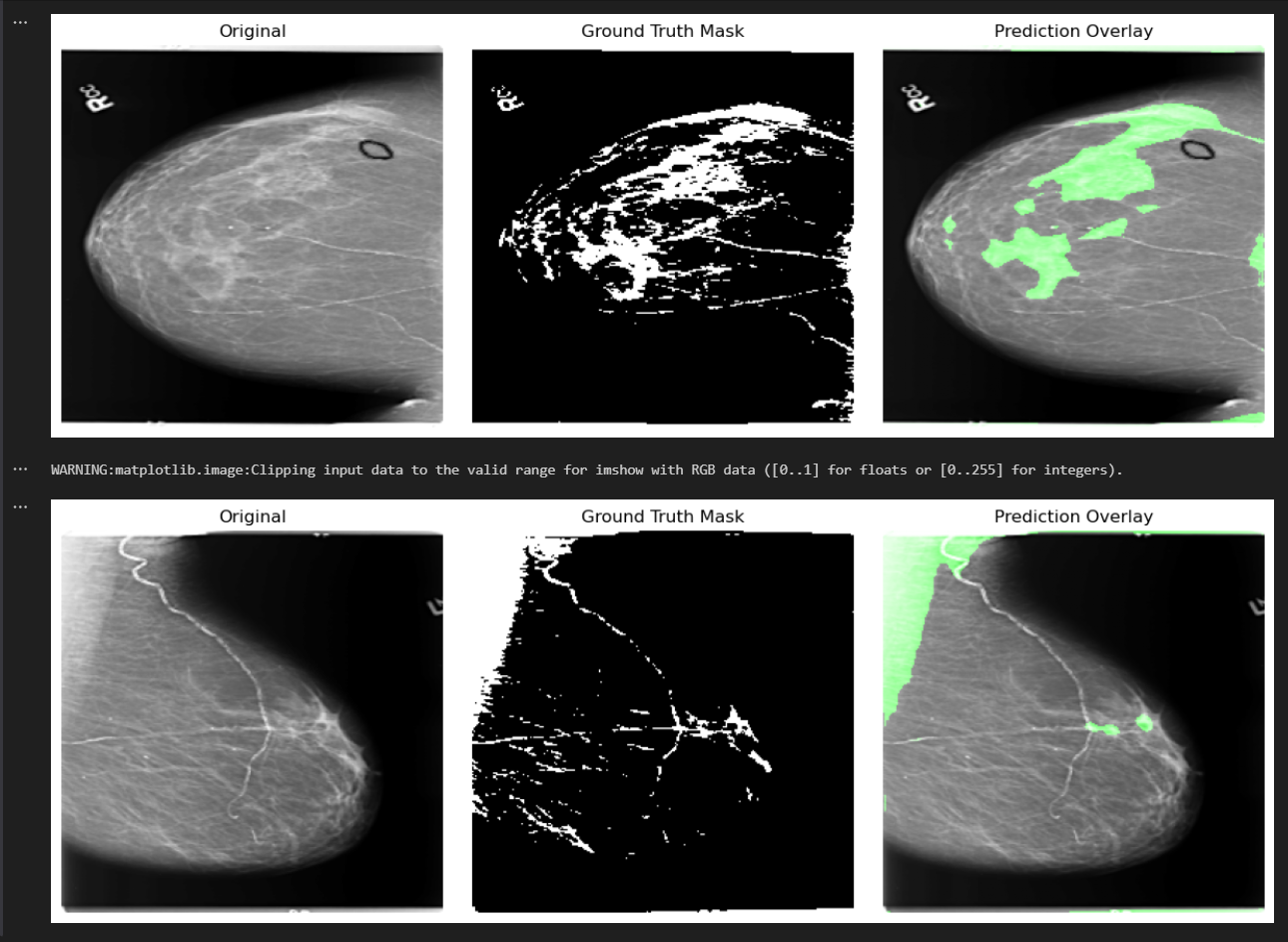
****

****

****

****

****

****

**Results:**

* CNN achieved high classification accuracy in distinguishing tumor types.
* U-Net successfully segmented tumor regions with high overlap with ground truth.
* The models demonstrated strong performance and potential applicability in medical imaging.

**References:**

1. Ronneberger, O., Fischer, P., & Brox, T. (2015). U-Net: Convolutional Networks for Biomedical Image Segmentation.
2. Breast Ultrasound Images Dataset on Kaggle.
3. Deep Learning with Python by François Chollet.
4. TensorFlow and Keras Documentation.