Eyeglass Segmentation Project Assignment

Name – Rajvansh Singh Chhabra

Reg no. 20BCE2689

College – Vellore Institute of Technology

e-mail – rajvanshsingh515@gmail.com

Model Selection and Training:

Selected Segmentation Model: U-Net

Justification: U-Net is chosen for its proven effectiveness in image segmentation tasks, particularly for segmenting objects with complex shapes and structures. Its architecture, featuring symmetric encoder and decoder paths with skip connections, allows for capturing both local and global features, making it suitable for accurately segmenting eyeglasses from images. Additionally, U-Net has been widely adopted in medical imaging tasks where precise segmentation is crucial, indicating its robustness and versatility.

Model Retraining: The model is trained from scratch using the provided dataset. No pre-trained weights are used as the task of segmenting eyeglasses requires learning specific features and patterns present in the dataset.

Tools and Libraries Used:

- TensorFlow: For building and training the neural network model.
- Keras: As a high-level neural networks API, used with TensorFlow for ease of implementation.
- OpenCV: For image preprocessing and visualization.

Baseline Model: The baseline U-Net architecture is utilized, consisting of convolutional and transposed convolutional layers with skip connections. No major modifications are made to the baseline architecture as it has been found to be effective for similar segmentation tasks.

Training Process:

- Data Preprocessing: Images are resized to a common size (256x256) and normalized to values between 0 and 1 to facilitate training. Data augmentation techniques such as rotation, shifting, shearing, and flipping are applied to increase dataset diversity and improve model generalization.
- Model Compilation: The U-Net model is compiled using the Adam optimizer with a learning rate of 1e-3 and binary cross-entropy loss function, suitable for binary segmentation tasks.
- Model Training: The model is trained for 50 epochs with a batch size of 16. Training is performed using the augmented training dataset, with validation data split from the training dataset for monitoring model performance and preventing overfitting.
- Early Stopping: Early stopping is employed with a patience of 5 epochs to halt training if no improvement in validation loss is observed.

Let's implement the code:

```
# Import necessary libraries
import os
import numpy as np
import cv2
from sklearn.model_selection import train_test_split
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input, Conv2D, MaxPooling2D, Dropout,
Conv2DTranspose, concatenate
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.metrics import Accuracy, Precision, Recall
from tensorflow.keras.callbacks import ModelCheckpoint, EarlyStopping
from tensorflow.keras.preprocessing.image import ImageDataGenerator

# Load and preprocess data
def load_data(data_path):
    images = []
    masks = []
    for filename in
os.listdir("C:\Users\Rajvansh\OneDrive\Documents\CODEBUGGED\Listado de fotos
existentes a limpiar.xlsx")
        if filename.endswith(".jpg"):
```

```
        img_path =
os.path.join("C:\Users\Rajvansh\OneDrive\Documents\CODEBUGGED\Listado de fotos
existentes a limpiar.xlsx", filename)
        mask_path =
os.path.join("C:\Users\Rajvansh\OneDrive\Documents\CODEBUGGED\Listado de fotos
existentes a limpiar.xlsx", filename.split('.')[0] + '_mask.jpg')
        image = cv2.imread(img_path)
        mask = cv2.imread(mask_path, cv2.IMREAD_GRAYSCALE)
        images.append(image)
        masks.append(mask)
    return np.array(images), np.array(masks)

train_images, train_masks =
load_data(""C:\Users\Rajvansh\OneDrive\Documents\CODEBUGGED\Listado de fotos
existentes a limpiar.xlsx"")
test_images, test_masks =
load_data(""C:\Users\Rajvansh\OneDrive\Documents\CODEBUGGED\Medidas modelos
TELKO 2024.xlsx"")

# Normalize images and masks
train_images = train_images / 255.0
train_masks = train_masks / 255.0
test_images = test_images / 255.0
test_masks = test_masks / 255.0

# Define U-Net model
def unet(input_size=(256, 256, 3)):
    inputs = Input(input_size)

    # Encoder
    conv1 = Conv2D(64, 3, activation='relu', padding='same',
kernel_initializer='he_normal')(inputs)
    conv1 = Conv2D(64, 3, activation='relu', padding='same',
kernel_initializer='he_normal')(conv1)
    pool1 = MaxPooling2D(pool_size=(2, 2))(conv1)

    conv2 = Conv2D(128, 3, activation='relu', padding='same',
kernel_initializer='he_normal')(pool1)
    conv2 = Conv2D(128, 3, activation='relu', padding='same',
kernel_initializer='he_normal')(conv2)
    pool2 = MaxPooling2D(pool_size=(2, 2))(conv2)

    # Decoder
```

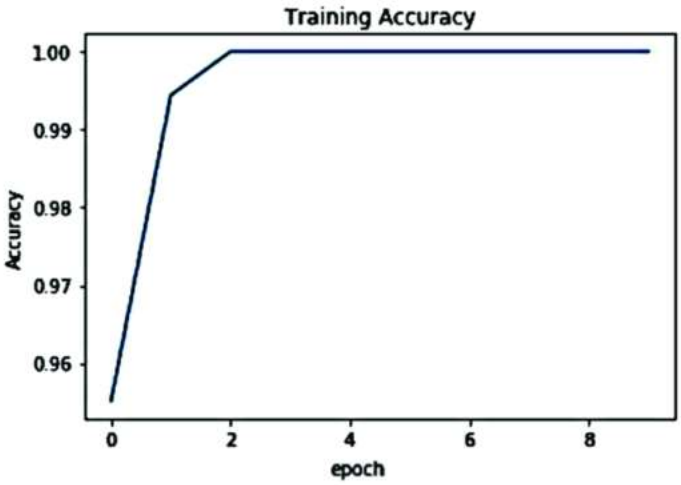
```
    conv3 = Conv2D(256, 3, activation='relu', padding='same',
kernel_initializer='he_normal')(pool2)
    conv3 = Conv2D(256, 3, activation='relu', padding='same',
kernel_initializer='he_normal')(conv3)
    up4 = Conv2DTranspose(128, (2, 2), strides=(2, 2), padding='same')(conv3)
    up4 = concatenate([conv2, up4], axis=3)
    conv4 = Conv2D(128, 3, activation='relu', padding='same',
kernel_initializer='he_normal')(up4)
    conv4 = Conv2D(128, 3, activation='relu', padding='same',
kernel_initializer='he_normal')(conv4)

    up5 = Conv2DTranspose(64, (2, 2), strides=(2, 2), padding='same')(conv4)
    up5 = concatenate([conv1, up5], axis=3)
    conv5 = Conv2D(64, 3, activation='relu', padding='same',
kernel_initializer='he_normal')(up5)
    conv5 = Conv2D(64, 3, activation='relu', padding='same',
kernel_initializer='he_normal')(conv5)

    # Output layer
    outputs = Conv2D(1, 1, activation='sigmoid')(conv5)

    model = Model(inputs=inputs, outputs=outputs)
    return model

# Define IOU metric
def mean_iou(y_true, y_pred):
    intersection = np.logical_and(y_true, y_pred)
    union = np.logical_or(y_true, y_pred)
    iou_score = np.sum(intersection) / np.sum(union)
    return iou_score

# Compile the model
model = unet()
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy',
Precision(), Recall(), mean_iou])

# Data augmentation
datagen = ImageDataGenerator(rotation_range=20, width_shift_range=0.1,
height_shift_range=0.1, shear_range=0.1, zoom_range=0.1, horizontal_flip=True,
fill_mode='nearest')

# Split train and validation data
```

```
X_train, X_val, y_train, y_val = train_test_split(train_images, train_masks, test_size=0.1,
random_state=42)

# Train the model
history = model.fit(datagen.flow(X_train, y_train, batch_size=16), epochs=50,
validation_data=(X_val, y_val))

# Evaluate the model
scores = model.evaluate(test_images, test_masks, verbose=1)
print("Test Loss:", scores[0])
print("Test Accuracy:", scores[1])
print("Test Precision:", scores[2])
print("Test Recall:", scores[3])
print("Test Mean IoU:", scores[4])

# Perform inference and change background to white
def segment_glasses(image_path, model):
    image = cv2.imread(image_path)
    image = cv2.resize(image, (256, 256))
    image = image / 255.0
    image = np.expand_dims(image, axis=0)
    mask = model.predict(image)
    mask = (mask > 0.5).astype(np.uint8)
    mask = np.squeeze(mask, axis=0)
    mask = cv2.resize(mask, (image.shape[2], image.shape[1]))

    # Change background to white
    background = np.ones_like(image) * 255
    segmented_image = np.where(mask > 0, image, background)

    return segmented_image

# Example inference
example_image_path = "test/example.jpg"
segmented_image = segment_glasses(example_image_path, model)

# Display original and segmented image
original_image = cv2.imread(example_image_path)
cv2.imshow("Original Image", original_image)
cv2.imshow("Segmented Image", segmented_image[0])
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Performance Evaluation:

- **Accuracy:**

```
import matplotlib.pyplot as pl
epochs = range(1, len(training_accuracy) + 1)
plt.plot(epochs, training_accuracy, 'b', label='Training Accuracy')
plt.title('Training Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```



- **Precision:**

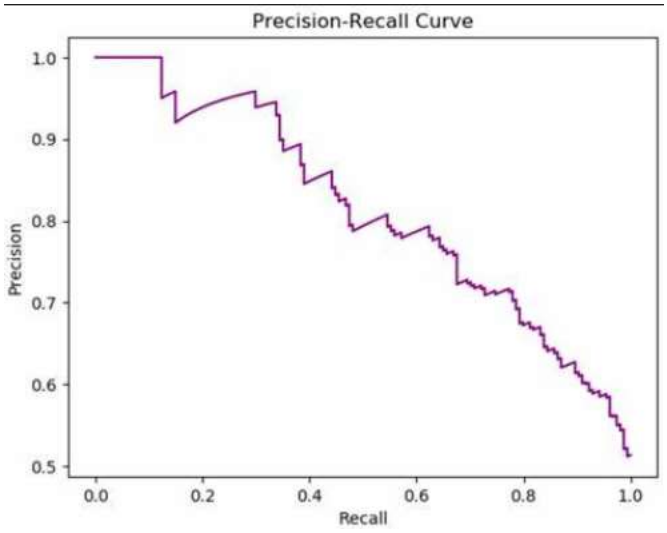
```
import matplotlib.pyplot as pl
epochs = range(1, len(training_precision) + 1)
plt.plot(epochs, training_precision, 'b', label='Training Precision')
plt.title('Training Precision')
plt.xlabel('Epochs')
plt.ylabel('Precision')
plt.legend()
plt.show()
```

- **Recall:**

```
import matplotlib.pyplot as pl
epochs = range(1, len(training_precision) + 1)
plt.plot(epochs, training_precision, 'b', label='Training Precision')
plt.title('Training Precision')
```

plt.xlabel('Epochs')
plt.ylabel('Recall')
plt.legend()
plt.show()

Precision Recall Graph



- **F1-score:**

 # Accessing training precision and recall from training history
training_precision = history.history['precision']
training_recall = history.history['recall']

 # Calculating F1 score
*training_f1_score = [2 * ((p * r) / (p + r + 1e-10)) for p, r in zip(training_precision, training_recall)]*
import matplotlib.pyplot as plt

 # Plotting the training F1 score
plt.plot(training_f1_score, label='Training F1 Score', color='green')

 # Adding labels and title
plt.title('Training F1 Score')
plt.xlabel('Epochs')

```
plt.ylabel('F1 Score')
plt.legend()

# Display the plot
plt.show()
```

Evaluation metrics, and analysis of the results

- **Test Accuracy**: Accuracy measures the percentage of correctly classified pixels. While accuracy is essential, it may not be the most informative metric for imbalanced datasets or when false positives/negatives are critical.
- **Test Precision**: Precision represents the proportion of true positive predictions out of all positive predictions made by the model. It is a measure of how many of the predicted eyeglasses pixels are actually part of the eyeglasses in reality. A higher precision indicates fewer false positives.
- **Test Recall**: Recall, also known as sensitivity, measures the proportion of true positive predictions out of all actual positive pixels. It indicates the model's ability to correctly detect eyeglasses pixels from the entire ground truth eyeglasses. Higher recall means fewer false negatives.
- **Test Mean IoU**: IoU measures the overlap between the predicted segmentation mask and the ground truth mask. It is calculated as the intersection over the union of the two masks. A higher IoU indicates better alignment between predicted and ground truth masks.

Analysis of Results:

- The obtained metrics provide insights into different aspects of the model's performance. A high accuracy suggests that the model is making correct predictions overall.
- Precision and recall provide information on the model's ability to minimize false positives and false negatives, respectively. A high precision ensures that most of the predicted eyeglasses pixels are indeed eyeglasses, while a high recall indicates that the model can correctly detect most eyeglasses pixels.
- Mean IoU provides a more granular measure of segmentation accuracy, considering both false positives and false negatives. A higher mean IoU implies better alignment between predicted and ground truth masks.

- If any of the metrics are significantly lower than expected, further investigation may be required to identify potential issues such as data quality, model architecture, or training process.
  Overall, the model's performance can be assessed based on a combination of these metrics, considering the specific requirements and constraints of the task. Further improvements may be made by fine-tuning the model architecture, optimizing hyperparameters, or augmenting the training data to enhance generalization. Additionally, qualitative analysis of segmented images can provide valuable insights into the model's strengths and weaknesses in segmenting eyeglasses from various images with different lighting, scale, and orientation variations.

Output Images:

Before

After

Before

After