Ensemble Approach for Blood Vessel Segmentation in Images: Comparing UNet and SegNet Models

Software and Hardware used:

Software:

- Python 3.12
- CUDA 12.3
- Imageio 2.35.1
- Joblib 1.4.2
- Keras 3.5.0
- Matplotlib 3.9.2
- Numpy 1.26.4
- OpenCV-Python 4.10.0.84
- Pandas 2.2.2
- Scikit-learn 1.5.1
- TensorFlow 2.17.0

Hardware:

- Intel i7 11800h
- RTX 3070 Laptop GPU

Code Snippets:

```python
for idx, (x, y) in tqdm(enumerate(zip(images, masks)), total = len(images)):     • "idx" is not accessed
    #print(x, y)
    name = os.path.basename(x).split(".")[0]
    x = cv2.imread(x, cv2.IMREAD_COLOR)
    y = imageio.mimread(y)[0]

    if augment == True:
        aug = HorizontalFlip(p = 1.0)
        augmented = aug(image = x, mask = y)
        x1 = augmented["image"]
        y1 = augmented["mask"]

        aug = VerticalFlip(p = 1.0)
        augmented = aug(image = x, mask = y)
        x2 = augmented["image"]
        y2 = augmented["mask"]

        aug = ElasticTransform(p = 1.0, alpha = 120, sigma = 120 * 0.05, alpha_affline = 120 * 0.03)
        augmented = aug(image = x, mask = y)
        x3 = augmented["image"]
        y3 = augmented["mask"]

        aug = GridDistortion(p = 1.0)
        augmented = aug(image = x, mask = y)
        x4 = augmented["image"]
        y4 = augmented["mask"]

        aug = OpticalDistortion(p = 1.0, distort_limit = 2, shift_limit = 0.5)
        augmented = aug(image = x, mask = y)
        x5 = augmented["image"]
        y5 = augmented["mask"]

        X = [x, x1, x2, x3, x4, x5]
        Y = [y, y1, y2, y3, y4, y5]

    else:
        X = [x]
        Y = [y]

    index = 0

    for i, m in zip(X, Y):
        i = cv2.resize(i, (w, h))
        m = cv2.resize(m, (w, h))

        if len(X) == 1:
            tmp_image_name = f"{name}.jpg"
            tmp_mask_name = f"{name}.jpg"
```

Data Augmentation

```python
def conv_block(x, num_filters):
    x = Conv2D(num_filters, (3, 3), padding='same', activation='relu')(x)
    x = Conv2D(num_filters, (3, 3), padding='same', activation='relu')(x)
    return x

def encoder_block(x, num_filters):
    x = conv_block(x, num_filters)
    p = MaxPooling2D((2, 2))(x)
    return x, p

def decoder_block(x, skip_features, num_filters):
    x = Conv2DTranspose(num_filters, (2, 2), strides=(2, 2), padding='same')(x)
    x = Concatenate()([x, skip_features])
    x = conv_block(x, num_filters)
    return x

def build_segnet(input_shape):
    inputs = Input(shape=input_shape)

    s1, p1 = encoder_block(inputs, 64)
    s2, p2 = encoder_block(p1, 128)
    s3, p3 = encoder_block(p2, 256)
    s4, p4 = encoder_block(p3, 512)

    b1 = conv_block(p4, 1024)

    d1 = decoder_block(b1, s4, 512)
    d2 = decoder_block(d1, s3, 256)
    d3 = decoder_block(d2, s2, 128)
    d4 = decoder_block(d3, s1, 64)

    outputs = Conv2D(1, (1, 1), activation='sigmoid')(d4)

    model = Model(inputs, outputs, name='SegNet')
    return model
```

Segnet Architecture

```
    train_x, train_y = load_data(train_path)
    train_x, train_y = shuffling(train_x, train_y)
    valid_x, valid_y = load_data(valid_path)

    print(f"train: {len(train_x)} - {len(train_y)}")
    print(f"valid: {len(valid_x)} - {len(valid_y)}")

    train_dataset = tf_dataset(train_x, train_y, batch_size=batch_size)
    valid_dataset = tf_dataset(valid_x, valid_y, batch_size=batch_size)

    train_steps = len(train_x) // batch_size
    valid_steps = len(valid_x) // batch_size

    if len(train_x) % batch_size != 0:
        train_steps += 1
    if len(valid_x) % batch_size != 0:
        valid_steps += 1

    model = build_segnet((h, w, 3))
    model.compile(
        loss=BinaryCrossentropy(),
        optimizer=Adam(learning_rate=lr),
        metrics=[Recall(), Precision()]
    )

    model.summary()

    callbacks = [
        ModelCheckpoint(model_path, verbose=1, save_best_only=True),
        ReduceLROnPlateau(monitor="val_loss", factor=0.1, patience=5, min_lr=1e-6, verbose=1),
        CSVLogger(csv_path),
        TensorBoard(),
        EarlyStopping(monitor="val_loss", patience=10, restore_best_weights=True)
    ]

    model.fit(
        train_dataset,
        epochs=num_epoch,
        validation_data=valid_dataset,
        steps_per_epoch=train_steps,
        validation_steps=valid_steps,
        callbacks=callbacks
    )
```

Segnet Training

```python
create_dir("segnet_results")

model = build_segnet((512, 512, 3))

try:
    with CustomObjectScope({"iou": iou, "dice_coef": dice_coef, "dice_loss": dice_loss}):
        model.load_weights("files/segnet_model.keras")
    print("Model loaded successfully.")
except Exception as e:
    print(f"Error loading model: {e}")
    exit()

dataset_path = os.path.join("new_data/test")

test_x, test_y = load_data(dataset_path)

SCORE = []

for x, y in tqdm(zip(test_x, test_y), total=len(test_x)):
    name = os.path.basename(x).split(".")[0]

    ori_x, x = read_image(x)
    ori_y, y = read_mask(y)

    y_pred = model.predict(np.expand_dims(x, axis=0))[0]
    y_pred = y_pred > 0.5
    y_pred = y_pred.astype(np.int32)
    y_pred = np.squeeze(y_pred, axis=-1)

    save_image_path = f"segnet_results/{name}.png"
    save_results(ori_x, ori_y, y_pred, save_image_path)

    y = y.flatten()
    y_pred = y_pred.flatten()

    acc_value = accuracy_score(y, y_pred)
    f1_value = f1_score(y, y_pred, labels=[0, 1], average="macro")
    jac_value = jaccard_score(y, y_pred, labels=[0, 1], average="macro")
    recall_value = recall_score(y, y_pred, labels=[0, 1], average="macro")
    precision_value = precision_score(y, y_pred, labels=[0, 1], average="macro")
    SCORE.append([name, acc_value, f1_value, jac_value, recall_value, precision_value])

score = [s[1:] for s in SCORE]
score = np.mean(score, axis=0)
```

Segnet evaluation

```python
def conv_block(inputs, num_filters):
    x = Conv2D(num_filters, 3, padding = "same")(inputs)
    x = BatchNormalization()(x)
    x = Activation("relu")(x)

    x = Conv2D(num_filters, 3, padding = "same")(x)
    x = BatchNormalization()(x)
    x = Activation("relu")(x)

    return x

def encoder_block(inputs, num_filters):
    x = conv_block(inputs, num_filters)
    p = MaxPool2D((2, 2))(x)
    return x, p

def decoder_block(inputs, skip_features, num_filters):
    x = Conv2DTranspose(num_filters, (2, 2), strides = 2, padding = "same")(inputs)
    x = Concatenate()([x, skip_features])
    x = conv_block(x, num_filters)
    return x

def build_unet(input_shape):
    inputs = Input(input_shape)

    s1, p1 = encoder_block(inputs, 64)
    s2, p2 = encoder_block(p1, 128)
    s3, p3 = encoder_block(p2, 256)
    s4, p4 = encoder_block(p3, 512)

    b1 = conv_block(p4, 1024)

    d1 = decoder_block(b1, s4, 512)
    d2 = decoder_block(d1, s3, 256)
    d3 = decoder_block(d2, s2, 128)
    d4 = decoder_block(d3, s1, 64)

    outputs = Conv2D(1, 1, padding = "same", activation = "sigmoid")(d4)

    model = Model(inputs, outputs, name = "UNET")

    return model
```

UNet Model

```python
batch_size = 2
lr = 1e-4
num_epoch = 100
model_path = os.path.join("files", "unet_model_v2.keras")
csv_path = os.path.join("files", "unet_data_train.csv")

dataset_path = "new_data"
train_path = os.path.join(dataset_path, "train")
valid_path = os.path.join(dataset_path, "test")

train_x, train_y = load_data(train_path)
train_x, train_y = shuffling(train_x, train_y)
valid_x, valid_y = load_data(valid_path)

print(f"train: {len(train_x)} - {len(train_y)}")
print(f"valid: {len(valid_x)} - {len(valid_y)}")

train_dataset = tf_dataset(train_x, train_y, batch_size=batch_size)
valid_dataset = tf_dataset(valid_x, valid_y, batch_size=batch_size)

train_steps = len(train_x) // batch_size
valid_steps = len(valid_x) // batch_size

if len(train_x) % batch_size != 0:
    train_steps += 1
if len(valid_x) % batch_size != 0:
    valid_steps += 1

model = build_unet((h, w, 3))
model.compile(loss=dice_loss, optimizer=Adam(learning_rate=lr), metrics=[dice_coef, iou, Recall(), Precision()])

model.summary()

callbacks = [
    ModelCheckpoint(model_path, verbose=1, save_best_only=True),
    ReduceLROnPlateau(monitor="val_loss", factor=0.1, patience=5, min_lr=1e-6, verbose=1),
    CSVLogger(csv_path),
    TensorBoard(),
    EarlyStopping(monitor="val_loss", patience=10, restore_best_weights=True)
]

model.fit(
    train_dataset,
    epochs=num_epoch,
    validation_data=valid_dataset,
    steps_per_epoch=train_steps,
    validation_steps=valid_steps,
    callbacks=callbacks
)
```

Unet Training

```python
model = build_unet((512, 512, 3))

try:
    with CustomObjectScope({"iou": iou, "dice_coef": dice_coef, "dice_loss": dice_loss}):
        model.load_weights("files/unet_model.h5")
    print("Model loaded successfully.")
except Exception as e:
    print(f"Error loading model: {e}")
    exit()

dataset_path = os.path.join("new_data/test")

test_x, test_y = load_data(dataset_path)

SCORE = []

for x, y in tqdm(zip(test_x, test_y), total=len(test_x)):
    name = os.path.basename(x).split(".")[0]

    ori_x, x = read_image(x)
    ori_y, y = read_mask(y)

    y_pred = model.predict(np.expand_dims(x, axis=0))[0]
    y_pred = y_pred > 0.5
    y_pred = y_pred.astype(np.int32)
    y_pred = np.squeeze(y_pred, axis=-1)

    save_image_path = f"unet_results/{name}.png"
    save_results(ori_x, ori_y, y_pred, save_image_path)

    y = y.flatten()
    y_pred = y_pred.flatten()

    acc_value = accuracy_score(y, y_pred)
    f1_value = f1_score(y, y_pred, labels=[0, 1], average="macro")
    jac_value = jaccard_score(y, y_pred, labels=[0, 1], average="macro")
    recall_value = recall_score(y, y_pred, labels=[0, 1], average="macro")
    precision_value = precision_score(y, y_pred, labels=[0, 1], average="macro")
    SCORE.append([name, acc_value, f1_value, jac_value, recall_value, precision_value])

score = [s[1:] for s in SCORE]
score = np.mean(score, axis=0)
```

Unet evaluation

```python
def train_logistic_regression_model(train_x, train_y, unet_model, segnet_model, save_path):
    X_train = []
    y_train = []

    for x, y in tqdm(zip(train_x, train_y), total=len(train_x)):
        _, x = read_image(x)
        _, y = read_mask(y)

        unet_pred = unet_model.predict(np.expand_dims(x, axis=0))[0]
        segnet_pred = segnet_model.predict(np.expand_dims(x, axis=0))[0]

        unet_pred = unet_pred.flatten().reshape(-1, 1)
        segnet_pred = segnet_pred.flatten().reshape(-1, 1)

        ensemble_input = np.concatenate([unet_pred, segnet_pred], axis=1)
        X_train.append(ensemble_input)
        y_train.append(y.flatten())

    X_train = np.concatenate(X_train, axis=0)
    y_train = np.concatenate(y_train, axis=0)

    logistic_regression_model = LogisticRegression()
    logistic_regression_model.fit(X_train, y_train)

    joblib.dump(logistic_regression_model, save_path)
    print(f"Logistic Regression model saved to {save_path}")

    return logistic_regression_model
```

Logistic Regression Ensemble training

Results:

	Accuracy	Dice Coefficient	Sensitivity(Recall)	F1 Score	Jaccard Score	Precision
UNet	0.94345	0.74280	0.92123	0.74280	0.64489	0.68177
SegNet	0.94545	0.75052	0.93091	0.75052	0.65251	0.68582
Logistic Regression Ensemble	0.97021	0.76775	0.76906	0.76775	0.67559	0.77251