nhndc5tjx

April 11, 2025

1 EE655 | ASSIGNMENT 2

2 ARYAVART | 230223

2.1 Description

(Note: I have done the question 4 in the end after question 5 and that too with again reintroducing part a and part c of ques 1 beacuse I had to create a separate jupyter notebook for this question as I was not able to do it in the main notebook, please don't mind.)

Question 1: In part A, I started by extracting the foreground of MNIST digits using Otsus thresholding to create binary masks. Then in part B, I used OpenCV to find the minimum enclosing circle for each digit. Finally, for Part C I made a new dataset by randomly picking 4 digits and combining their images and masks into a 2x2 grid.

Question 2: I built a CNN model to learn how to extract the foreground from digit images. It was trained on the original and masked images and gave a test IoU of 0.9353, which shows that the predicted masks were very close to the ground truth.

Question 3: I created a dual-output CNN that could classify the digit and also predict the enclosing circle's position and size. It achieved 98.95% classification accuracy and the circle predictions were very accurate too, with a regression loss of just 0.0006. When classification was correct, the predicted circles matched well with the true ones, with an IoU of 0.8715.

Question 4: I trained a deep learning model for semantic segmentation on 2×2 MNIST grid images using 10-channel one-hot encoded masks. Foreground masks were generated using Otsu thresholding. The model effectively learned to segment and classify digits in each region. It achieved a Test Loss of 0.1193 and Dice Coefficient of 0.8807.

Question 5: I extracted a video and background image from a ZIP file and used OpenCV's MOG2 background subtractor to isolate moving objects. After cleaning the mask with morphological operations, I replaced the original background with a new image. The final video, showed the moving subject smoothly blended onto the new background.

[2]:
```
import torch
print(torch.cuda.get_device_name(0))
print(torch.cuda.device_count())
print(torch.cuda.current_device())
print(torch.cuda.get_device_capability(0))
```

```
Tesla T4
1
0
(7, 5)
```

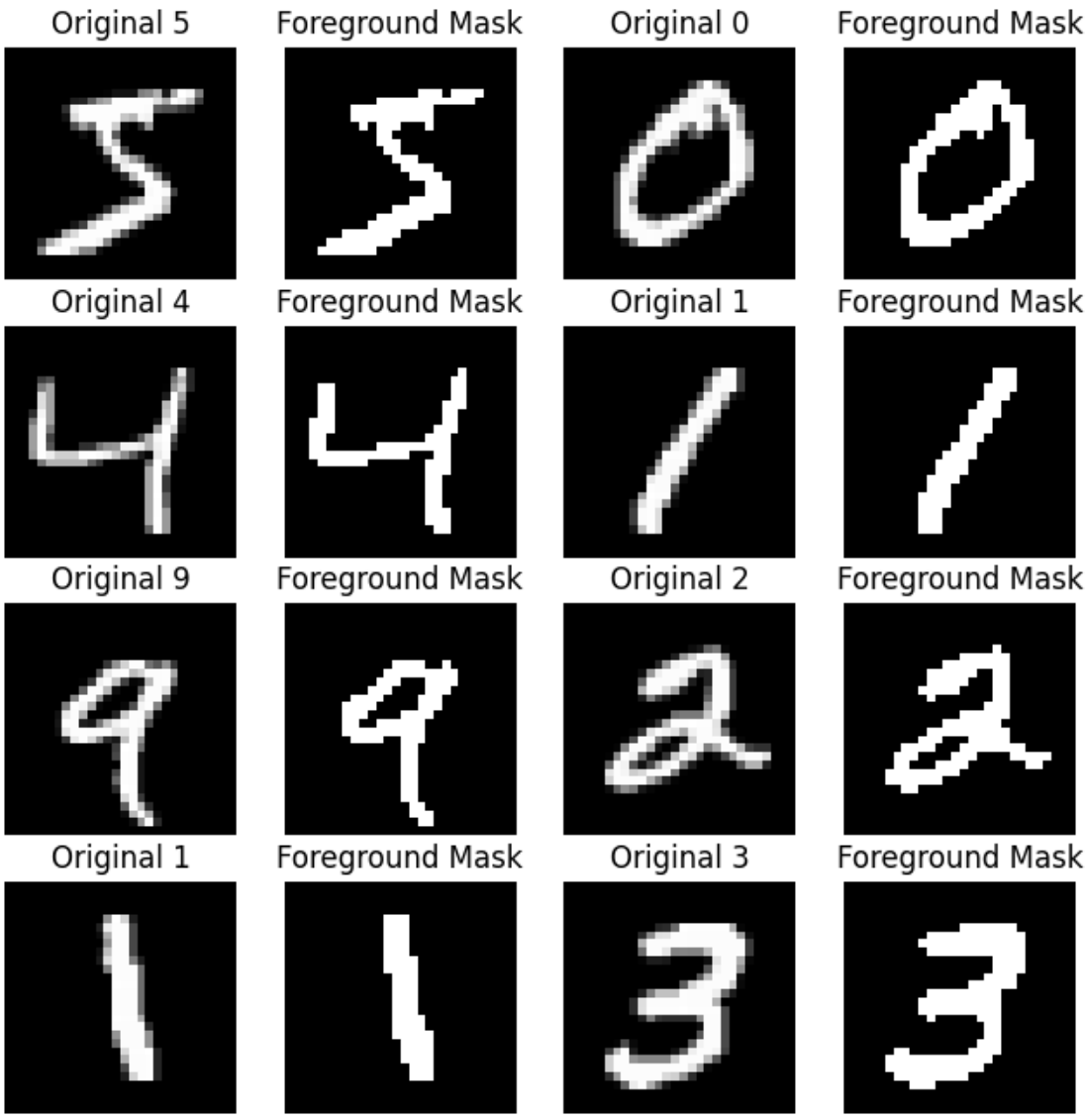
[3]:
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import fetch_openml
from skimage.filters import threshold_otsu
from skimage.util import montage
```

[4]:
```
mnist = fetch_openml('mnist_784', version=1, as_frame=False)
X = mnist.data.reshape(-1, 28, 28)
y = mnist.target.astype(np.uint8)
```

3 QUES 1 | PART A

[5]:
```
foreground_masks = np.zeros_like(X, dtype=np.uint8)

for i in range(len(X)):
    thresh = threshold_otsu(X[i])
    foreground_masks[i] = (X[i] > thresh).astype(np.uint8)
```

[6]:
```
num_images = 8
fig, axes = plt.subplots(4, 4, figsize=(8, 8))

for i in range(2 * num_images):
    ax = axes.flat[i]
    img_idx = i // 2
    if i % 2 == 0:
        ax.imshow(X[img_idx], cmap='gray')
        ax.set_title(f'Original {y[img_idx]}')
    else:
        ax.imshow(foreground_masks[img_idx], cmap='gray')
        ax.set_title('Foreground Mask')
    ax.axis('off')
```



4 QUES 1 | PART B

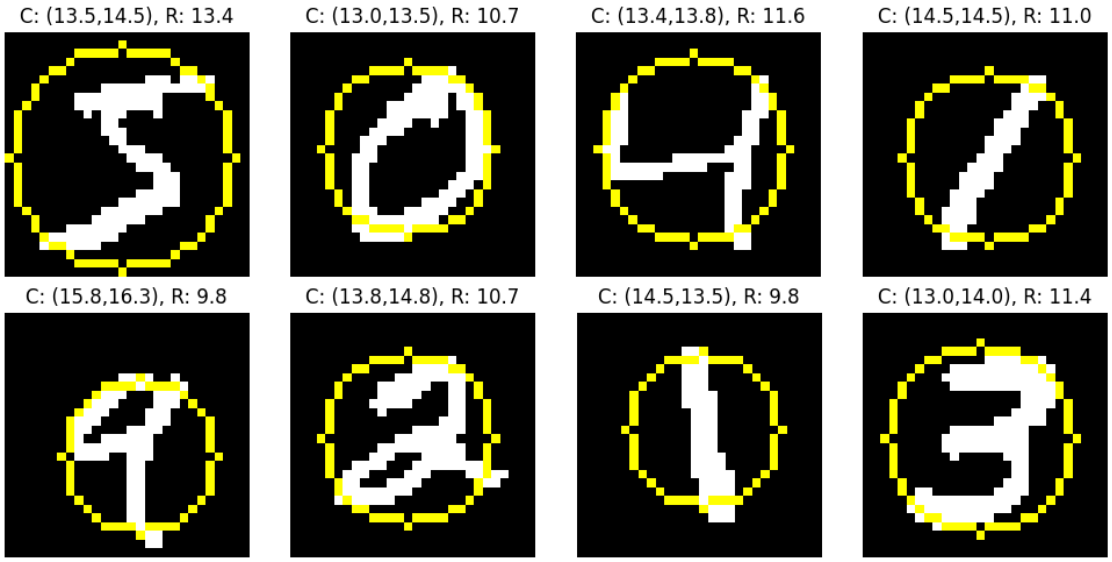
[7]:
```
from skimage.measure import regionprops
import cv2
```

[8]:
```
def get_min_enclosing_circle(mask):
    contours, _ = cv2.findContours(mask.astype(np.uint8), cv2.RETR_EXTERNAL,␣
 ↪cv2.CHAIN_APPROX_SIMPLE)
    if not contours:
        return None
    largest_contour = max(contours, key=cv2.contourArea)
```

```
    (x, y), radius = cv2.minEnclosingCircle(largest_contour)
    return (x, y), radius
```

[9]:
```
labels = []
circle_coords = []

for idx, mask in enumerate(foreground_masks):
    circle_info = get_min_enclosing_circle(mask)
    if circle_info:
        (xc, yc), radius = circle_info
    else:
        xc, yc, radius = 14.0, 14.0, 0.0

    labels.append(y[idx])
    circle_coords.append((xc / 28.0, yc / 28.0, radius / 28.0))
```

[10]:
```
fig, axes = plt.subplots(2, 4, figsize=(10, 5))
for idx, ax in enumerate(axes.flat):
    mask = foreground_masks[idx]
    circle_info = get_min_enclosing_circle(mask)

    rgb_image = np.stack([mask * 255] * 3, axis=-1).astype(np.uint8)

    if circle_info:
        (xc, yc), radius = circle_info
        cv2.circle(rgb_image,(int(xc), int(yc)),int(radius),color=(255, 255,␣
 ↪0),thickness=1)

    ax.imshow(rgb_image)
    if circle_info:
        ax.set_title(f'C: ({xc:.1f},{yc:.1f}), R: {radius:.1f}')
    else:
        ax.set_title(f'Class: {y[idx]} (No circle)')
    ax.axis('off')

plt.tight_layout()
plt.show()
```



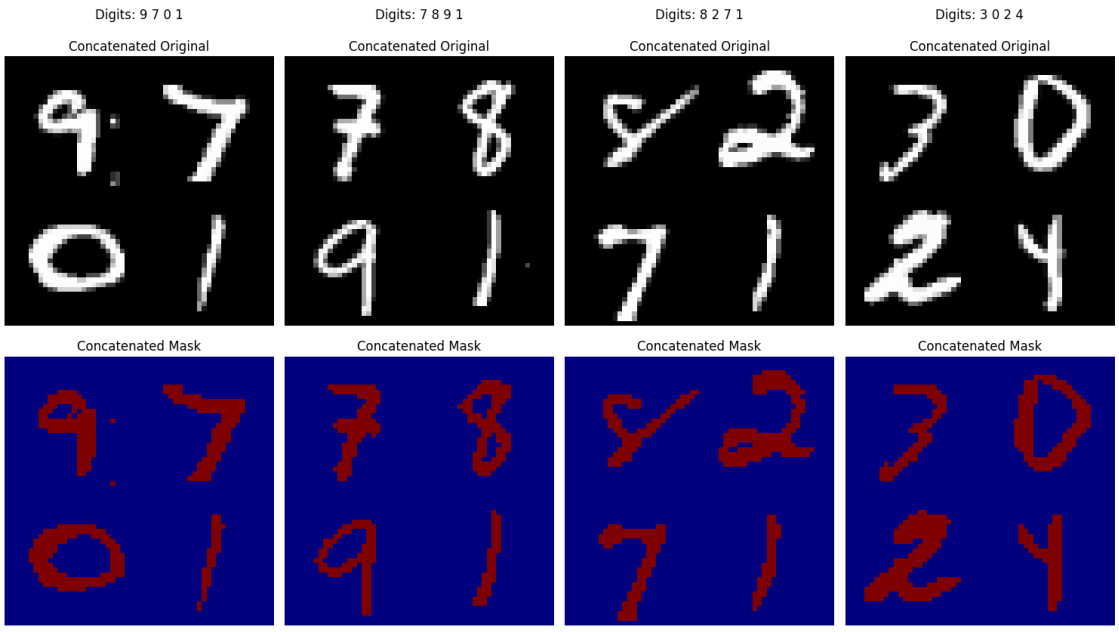
5 QUES 1 | PART C

[12]:
```python
def create_concatenated_dataset(images, masks, labels, num_samples=10000):
    concatenated_images = []
    concatenated_masks = []
    concatenated_labels = []

    for _ in range(num_samples):
        indices = np.random.choice(len(images), 4, replace=False)

        selected_images = [images[i] for i in indices]
        selected_masks = [masks[i] for i in indices]
        selected_labels = [int(labels[i]) for i in indices]

        top_row = np.hstack(selected_images[:2])
        bottom_row = np.hstack(selected_images[2:])
        concatenated_image = np.vstack([top_row, bottom_row])

        top_row_mask = np.hstack(selected_masks[:2])
        bottom_row_mask = np.hstack(selected_masks[2:])
        concatenated_mask = np.vstack([top_row_mask, bottom_row_mask])

        combined_label = " ".join(map(str, selected_labels))

        concatenated_images.append(concatenated_image)
        concatenated_masks.append(concatenated_mask)
        concatenated_labels.append(selected_labels)
```

```
    return (np.array(concatenated_images, dtype=np.uint8),np.
 ↪array(concatenated_masks, dtype=np.uint8),np.array(concatenated_labels,␣
 ↪dtype=np.int32))
```

[13]:
```
concat_images, concat_masks, concat_labels = create_concatenated_dataset(X,␣
 ↪foreground_masks, y, num_samples=10000)
```

[14]:
```
fig, axes = plt.subplots(2, 4, figsize=(15, 8))

for i in range(4):
    label_str = " ".join(str(digit) for digit in concat_labels[i])

    axes[0, i].imshow(concat_images[i], cmap='gray')
    axes[0, i].set_title(f'Digits: {label_str}\n\nConcatenated Original')
    axes[0, i].axis('off')

    axes[1, i].imshow(concat_masks[i], cmap='jet')
    axes[1, i].set_title('Concatenated Mask')
    axes[1, i].axis('off')

plt.suptitle('', y=1.05)
plt.tight_layout(rect=[0, 0, 1, 1.10])
plt.show()
```



6 QUES 2

[15]:
```
import tensorflow as tf
from tensorflow.keras import layers, models
from sklearn.model_selection import train_test_split
from tensorflow.keras.metrics import MeanIoU
```

[18]:
```
X_normalized = X / 255.0
X_train, X_test, y_train, y_test = train_test_split(X_normalized,␣
 ↪foreground_masks, test_size=0.2, random_state=42)
```

[19]:
```
X_train = np.expand_dims(X_train, axis=-1)
X_test = np.expand_dims(X_test, axis=-1)
y_train = np.expand_dims(y_train, axis=-1)
y_test = np.expand_dims(y_test, axis=-1)
```

[17]:
```
def build_foreground_model(input_shape=(28, 28, 1)):
    inputs = layers.Input(input_shape)

    conv1 = layers.Conv2D(32, 3, activation='relu', padding='same')(inputs)
    conv1 = layers.Conv2D(32, 3, activation='relu', padding='same')(conv1)
    pool1 = layers.MaxPooling2D(pool_size=(2, 2))(conv1)

    conv2 = layers.Conv2D(64, 3, activation='relu', padding='same')(pool1)
    conv2 = layers.Conv2D(64, 3, activation='relu', padding='same')(conv2)
    pool2 = layers.MaxPooling2D(pool_size=(2, 2))(conv2)

    conv3 = layers.Conv2D(128, 3, activation='relu', padding='same')(pool2)
    conv3 = layers.Conv2D(128, 3, activation='relu', padding='same')(conv3)

    up4 = layers.UpSampling2D(size=(2, 2))(conv3)
    up4 = layers.Conv2D(64, 2, activation='relu', padding='same')(up4)
    merge4 = layers.concatenate([conv2, up4], axis=3)
    conv4 = layers.Conv2D(64, 3, activation='relu', padding='same')(merge4)
    conv4 = layers.Conv2D(64, 3, activation='relu', padding='same')(conv4)

    up5 = layers.UpSampling2D(size=(2, 2))(conv4)
    up5 = layers.Conv2D(32, 2, activation='relu', padding='same')(up5)
    merge5 = layers.concatenate([conv1, up5], axis=3)
    conv5 = layers.Conv2D(32, 3, activation='relu', padding='same')(merge5)
    conv5 = layers.Conv2D(32, 3, activation='relu', padding='same')(conv5)

    outputs = layers.Conv2D(1, 1, activation='sigmoid')(conv5)

    model = models.Model(inputs=inputs, outputs=outputs)
    model.compile(optimizer='adam', loss='binary_crossentropy',␣
 ↪metrics=[MeanIoU(num_classes=2)])
```

```
    return model
```

[ ]:
```
foreground_model = build_foreground_model()

history = foreground_model.fit(X_train,␣
↪y_train,batch_size=64,epochs=15,validation_split=0.1,verbose=1)
```

```
Model: "functional"
```

Layer (type)	Output Shape	Param #	Connected to
input_layer (InputLayer)	(None, 28, 28, 1)	0	-
conv2d (Conv2D)	(None, 28, 28, 32)	320	input_layer[0][0]
conv2d_1 (Conv2D)	(None, 28, 28, 32)	9,248	conv2d[0][0]
max_pooling2d (MaxPooling2D)	(None, 14, 14, 32)	0	conv2d_1[0][0]
conv2d_2 (Conv2D)	(None, 14, 14, 64)	18,496	max_pooling2d[0]…
conv2d_3 (Conv2D)	(None, 14, 14, 64)	36,928	conv2d_2[0][0]
max_pooling2d_1 (MaxPooling2D)	(None, 7, 7, 64)	0	conv2d_3[0][0]
conv2d_4 (Conv2D)	(None, 7, 7, 128)	73,856	max_pooling2d_1[…
conv2d_5 (Conv2D)	(None, 7, 7, 128)	147,584	conv2d_4[0][0]
up_sampling2d (UpSampling2D)	(None, 14, 14, 128)	0	conv2d_5[0][0]
conv2d_6 (Conv2D)	(None, 14, 14, 64)	32,832	up_sampling2d[0]…
concatenate (Concatenate)	(None, 14, 14, 128)	0	conv2d_3[0][0], conv2d_6[0][0]
conv2d_7 (Conv2D)	(None, 14, 14,	73,792	concatenate[0][0]

```
                    64)

 conv2d_8 (Conv2D)     (None, 14, 14,          36,928   conv2d_7[0][0]
                       64)

 up_sampling2d_1       (None, 28, 28,               0   conv2d_8[0][0]
 (UpSampling2D)        64)

 conv2d_9 (Conv2D)     (None, 28, 28,           8,224   up_sampling2d_1[…
                       32)

 concatenate_1         (None, 28, 28,               0   conv2d_1[0][0],
 (Concatenate)         64)                              conv2d_9[0][0]

 conv2d_10 (Conv2D)    (None, 28, 28,          18,464   concatenate_1[0]…
                       32)

 conv2d_11 (Conv2D)    (None, 28, 28,           9,248   conv2d_10[0][0]
                       32)

 conv2d_12 (Conv2D)    (None, 28, 28, 1)           33   conv2d_11[0][0]


 Total params: 465,953 (1.78 MB)

 Trainable params: 465,953 (1.78 MB)

 Non-trainable params: 0 (0.00 B)

Epoch 1/15
788/788             29s 24ms/step -
loss: 0.0641 - mean_io_u: 0.4579 - val_loss: 0.0031 - val_mean_io_u: 0.8404
Epoch 2/15
788/788             28s 15ms/step -
loss: 0.0031 - mean_io_u: 0.8809 - val_loss: 0.0022 - val_mean_io_u: 0.9230
Epoch 3/15
788/788             11s 14ms/step -
loss: 0.0025 - mean_io_u: 0.9237 - val_loss: 0.0021 - val_mean_io_u: 0.9338
Epoch 4/15
788/788             21s 14ms/step -
loss: 0.0022 - mean_io_u: 0.9352 - val_loss: 0.0031 - val_mean_io_u: 0.9252
Epoch 5/15
788/788             21s 16ms/step -
loss: 0.0023 - mean_io_u: 0.9347 - val_loss: 0.0031 - val_mean_io_u: 0.9354
Epoch 6/15
788/788             21s 16ms/step -
```

```
loss: 0.0024 - mean_io_u: 0.9311 - val_loss: 0.0019 - val_mean_io_u: 0.9321
Epoch 7/15
788/788             12s 15ms/step -
loss: 0.0021 - mean_io_u: 0.9362 - val_loss: 0.0019 - val_mean_io_u: 0.9386
Epoch 8/15
788/788             21s 16ms/step -
loss: 0.0022 - mean_io_u: 0.9365 - val_loss: 0.0022 - val_mean_io_u: 0.9402
Epoch 9/15
788/788             20s 15ms/step -
loss: 0.0022 - mean_io_u: 0.9330 - val_loss: 0.0022 - val_mean_io_u: 0.9414
Epoch 10/15
788/788             21s 16ms/step -
loss: 0.0020 - mean_io_u: 0.9353 - val_loss: 0.0020 - val_mean_io_u: 0.9315
Epoch 11/15
788/788             20s 15ms/step -
loss: 0.0019 - mean_io_u: 0.9369 - val_loss: 0.0021 - val_mean_io_u: 0.9414
Epoch 12/15
788/788             20s 15ms/step -
loss: 0.0019 - mean_io_u: 0.9382 - val_loss: 0.0026 - val_mean_io_u: 0.9419
Epoch 13/15
788/788             11s 15ms/step -
loss: 0.0019 - mean_io_u: 0.9372 - val_loss: 0.0019 - val_mean_io_u: 0.9346
Epoch 14/15
788/788             11s 14ms/step -
loss: 0.0018 - mean_io_u: 0.9369 - val_loss: 0.0025 - val_mean_io_u: 0.9423
Epoch 15/15
788/788             12s 15ms/step -
loss: 0.0018 - mean_io_u: 0.9388 - val_loss: 0.0017 - val_mean_io_u: 0.9351
```

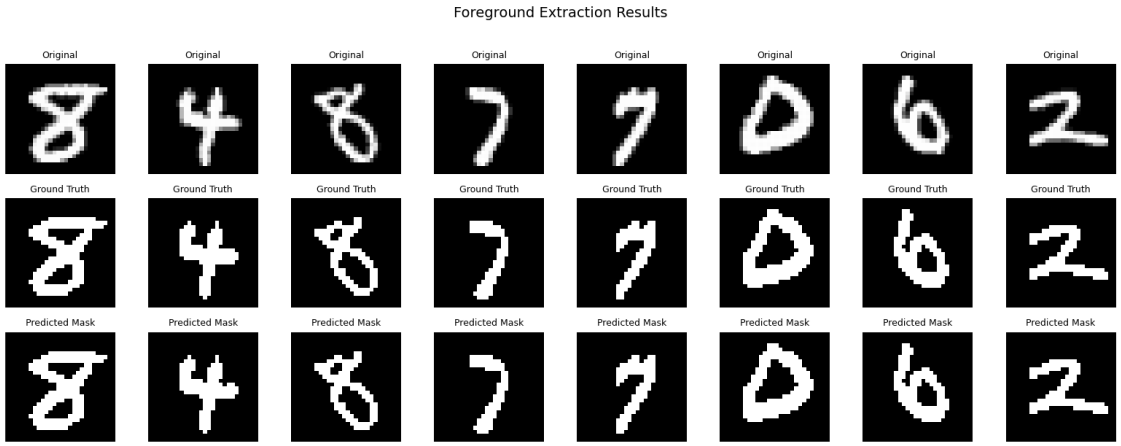
7 RESULTS

[19]:
```
test_results = foreground_model.evaluate(X_test, y_test, verbose=0)
print(f"\nTest Loss: {test_results[0]:.4f}")
print(f"Test IoU: {test_results[1]:.4f}")
```

```
Test Loss: 0.0017
Test IoU: 0.9353
```

[ ]:
```
fig, axes = plt.subplots(3, 8, figsize=(16, 6))

for i in range(8):
    axes[0, i].imshow(X_test[i, ..., 0], cmap='gray')
    axes[0, i].set_title('Original', fontsize=9)
    axes[0, i].axis('off')

    axes[1, i].imshow(y_test[i, ..., 0], cmap='gray')
```

```
    axes[1, i].set_title('Ground Truth', fontsize=9)
    axes[1, i].axis('off')

    axes[2, i].imshow(pred_masks[i, ..., 0], cmap='gray')
    axes[2, i].set_title('Predicted Mask', fontsize=9)
    axes[2, i].axis('off')

plt.suptitle('Foreground Extraction Results', fontsize=14, y=1.02)
plt.subplots_adjust(hspace=0.5)
plt.tight_layout()
plt.show()
```



8 QUES 3

[16]:
```
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.losses import CategoricalCrossentropy
```

[21]:
```
labels = np.array(labels)
circle_coords = np.array(circle_coords)

X_train_c, X_test_c, y_train_c, y_test_c, circle_train, circle_test =␣
 ↪train_test_split(X_normalized, labels, circle_coords, test_size=0.2,␣
 ↪random_state=42)

X_train_c = np.expand_dims(X_train_c, axis=-1)
X_test_c = np.expand_dims(X_test_c, axis=-1)

y_train_c = to_categorical(y_train_c, 10)
y_test_c = to_categorical(y_test_c, 10)
```

[ ]:
```
def build_circlization_model(input_shape=(28, 28, 1)):
    inputs = layers.Input(input_shape)

    x = layers.Conv2D(32, 3, activation='relu')(inputs)
    x = layers.MaxPooling2D(2)(x)
    x = layers.Conv2D(64, 3, activation='relu')(x)
    x = layers.MaxPooling2D(2)(x)
    x = layers.Conv2D(128, 3, activation='relu')(x)
    x = layers.Flatten()(x)
    x = layers.Dense(128, activation='relu')(x)

    class_output = layers.Dense(10, activation='softmax', name='class')(x)

    circle_output = layers.Dense(3, activation='sigmoid', name='circle')(x)

    model = models.Model(inputs=inputs, outputs=[class_output, circle_output])

    model.compile(optimizer='adam',loss={'class':␣
 ↪CategoricalCrossentropy(),'circle': 'mse'},loss_weights={'class': 1.0,␣
 ↪'circle': 0.5}, metrics={'class': 'accuracy'})

    return model
```

[23]:
```
circlization_model = build_circlization_model()

history = circlization_model.fit(X_train_c,{'class': y_train_c, 'circle':␣
 ↪circle_train},batch_size=64, epochs=15,validation_split=0.1,verbose=1)
```

```
Epoch 1/15
788/788             12s 7ms/step -
circle_loss: 0.0075 - class_accuracy: 0.8847 - class_loss: 0.3790 - loss: 0.3828
- val_circle_loss: 0.0026 - val_class_accuracy: 0.9848 - val_class_loss: 0.0451
- val_loss: 0.0466
Epoch 2/15
788/788             4s 5ms/step -
circle_loss: 0.0024 - class_accuracy: 0.9847 - class_loss: 0.0496 - loss: 0.0508
- val_circle_loss: 0.0019 - val_class_accuracy: 0.9868 - val_class_loss: 0.0391
- val_loss: 0.0403
Epoch 3/15
788/788             3s 4ms/step -
circle_loss: 0.0017 - class_accuracy: 0.9902 - class_loss: 0.0315 - loss: 0.0323
- val_circle_loss: 0.0017 - val_class_accuracy: 0.9875 - val_class_loss: 0.0403
- val_loss: 0.0413
Epoch 4/15
788/788             4s 4ms/step -
circle_loss: 0.0013 - class_accuracy: 0.9923 - class_loss: 0.0256 - loss: 0.0262
- val_circle_loss: 0.0010 - val_class_accuracy: 0.9889 - val_class_loss: 0.0391
```

```
- val_loss: 0.0398
Epoch 5/15
788/788            4s 5ms/step -
circle_loss: 9.8615e-04 - class_accuracy: 0.9951 - class_loss: 0.0165 - loss:
0.0170 - val_circle_loss: 8.0346e-04 - val_class_accuracy: 0.9886 -
val_class_loss: 0.0361 - val_loss: 0.0367
Epoch 6/15
788/788            4s 4ms/step -
circle_loss: 8.1095e-04 - class_accuracy: 0.9957 - class_loss: 0.0131 - loss:
0.0135 - val_circle_loss: 7.1162e-04 - val_class_accuracy: 0.9904 -
val_class_loss: 0.0319 - val_loss: 0.0324
Epoch 7/15
788/788            3s 4ms/step -
circle_loss: 7.2516e-04 - class_accuracy: 0.9961 - class_loss: 0.0115 - loss:
0.0119 - val_circle_loss: 7.2049e-04 - val_class_accuracy: 0.9905 -
val_class_loss: 0.0276 - val_loss: 0.0281
Epoch 8/15
788/788            3s 4ms/step -
circle_loss: 6.7929e-04 - class_accuracy: 0.9960 - class_loss: 0.0123 - loss:
0.0127 - val_circle_loss: 6.6408e-04 - val_class_accuracy: 0.9887 -
val_class_loss: 0.0393 - val_loss: 0.0389
Epoch 9/15
788/788            4s 5ms/step -
circle_loss: 6.8209e-04 - class_accuracy: 0.9976 - class_loss: 0.0082 - loss:
0.0085 - val_circle_loss: 6.7834e-04 - val_class_accuracy: 0.9907 -
val_class_loss: 0.0327 - val_loss: 0.0328
Epoch 10/15
788/788            3s 4ms/step -
circle_loss: 6.8236e-04 - class_accuracy: 0.9974 - class_loss: 0.0084 - loss:
0.0088 - val_circle_loss: 6.5576e-04 - val_class_accuracy: 0.9911 -
val_class_loss: 0.0334 - val_loss: 0.0340
Epoch 11/15
788/788            5s 5ms/step -
circle_loss: 6.7430e-04 - class_accuracy: 0.9974 - class_loss: 0.0081 - loss:
0.0085 - val_circle_loss: 6.9083e-04 - val_class_accuracy: 0.9916 -
val_class_loss: 0.0267 - val_loss: 0.0272
Epoch 12/15
788/788            4s 5ms/step -
circle_loss: 6.7745e-04 - class_accuracy: 0.9981 - class_loss: 0.0053 - loss:
0.0056 - val_circle_loss: 7.2571e-04 - val_class_accuracy: 0.9911 -
val_class_loss: 0.0327 - val_loss: 0.0332
Epoch 13/15
788/788            3s 4ms/step -
circle_loss: 6.7869e-04 - class_accuracy: 0.9980 - class_loss: 0.0062 - loss:
0.0066 - val_circle_loss: 6.6001e-04 - val_class_accuracy: 0.9916 -
val_class_loss: 0.0281 - val_loss: 0.0286
Epoch 14/15
788/788            6s 5ms/step -
```

```
circle_loss: 6.5847e-04 - class_accuracy: 0.9989 - class_loss: 0.0034 - loss:
0.0037 - val_circle_loss: 6.9962e-04 - val_class_accuracy: 0.9905 -
val_class_loss: 0.0404 - val_loss: 0.0410
Epoch 15/15
788/788             5s 4ms/step -
circle_loss: 6.6012e-04 - class_accuracy: 0.9984 - class_loss: 0.0051 - loss:
0.0054 - val_circle_loss: 6.3946e-04 - val_class_accuracy: 0.9900 -
val_class_loss: 0.0410 - val_loss: 0.0416
```

9 RESULTS

[24]:
```
test_results = circlization_model.evaluate(X_test_c,{'class': y_test_c,␣
 ↪'circle': circle_test},verbose=0)

print("\nTest Results:")
print(f"Classification Loss: {test_results[1]:.4f}")
print(f"Classification Accuracy: {test_results[3]:.4f}")
print(f"Circle Regression Loss: {test_results[2]:.4f}")
```

```
Test Results:
Classification Loss: 0.0360
Classification Accuracy: 0.9895
Circle Regression Loss: 0.0006
```

[29]:
```
def calculate_iou(true_circles, pred_circles, true_labels, pred_labels):
    ious = []
    for true, pred, true_l, pred_l in zip(true_circles, pred_circles,␣
 ↪true_labels, pred_labels):
        if np.argmax(true_l) != np.argmax(pred_l):
            ious.append(0.0)
            continue

        true_x, true_y = true[0] * 28, true[1] * 28
        true_r = true[2] * 28
        pred_x, pred_y = pred[0] * 28, pred[1] * 28
        pred_r = pred[2] * 28

        dist = np.sqrt((true_x - pred_x)**2 + (true_y - pred_y)**2)

        if dist > (true_r + pred_r):
            iou = 0.0
        else:
            r1, r2 = true_r, pred_r
            if r1 < r2:
                r1, r2 = r2, r1
```

```
            cos1 = np.clip((dist**2 + r2**2 - r1**2) / (2 * dist * r2), -1.0, 1.
↪0)
            cos2 = np.clip((dist**2 + r1**2 - r2**2) / (2 * dist * r1), -1.0, 1.
↪0)

            part1 = r2**2 * np.arccos(cos1)
            part2 = r1**2 * np.arccos(cos2)

            val = (-dist + r2 + r1)*(dist + r2 - r1)*(dist - r2 + r1)*(dist +␣
↪r2 + r1)
            part3 = 0.5 * np.sqrt(np.maximum(val, 0))

            intersection = part1 + part2 - part3
            union = np.pi * (r1**2 + r2**2) - intersection
            iou = intersection / union if union > 0 else 0.0

        ious.append(iou)

    return np.mean(ious)
```

10 IoU RESULT

[30]:
```
class_pred, circle_pred = circlization_model.predict(X_test_c)
mean_iou = calculate_iou(circle_test, circle_pred, y_test_c, class_pred)
print(f"\nTest IoU (with classification): {mean_iou:.4f}")
```

```
438/438             1s 1ms/step

Test IoU (with classification): 0.8715
```

[31]:
```
sample_indices = np.random.choice(len(X_test_c), 8, replace=False)

fig, axes = plt.subplots(3, 8, figsize=(20, 10))
for i, idx in enumerate(sample_indices):
    img = X_test_c[idx, ..., 0]
    true_class = np.argmax(y_test_c[idx])
    pred_class = np.argmax(class_pred[idx])

    true_x, true_y = circle_test[idx][0] * 28, circle_test[idx][1] * 28
    true_r = circle_test[idx][2] * 28

    pred_x, pred_y = circle_pred[idx][0] * 28, circle_pred[idx][1] * 28
    pred_r = circle_pred[idx][2] * 28

    axes[0, i].imshow(img, cmap='gray')
    axes[0, i].set_title(f'True: {true_class}\nPred: {pred_class}')
```

```
    axes[0, i].axis('off')

    axes[1, i].imshow(img, cmap='gray')
    circle = plt.Circle((true_x, true_y), true_r, color='red', fill=False,␣
 ↪linewidth=2)
    axes[1, i].add_patch(circle)
    axes[1, i].set_title('True Circle')
    axes[1, i].axis('off')

    axes[2, i].imshow(img, cmap='gray')
    circle = plt.Circle((pred_x, pred_y), pred_r, color='blue', fill=False,␣
 ↪linewidth=2)
    axes[2, i].add_patch(circle)
    axes[2, i].set_title('Predicted Circle')
    axes[2, i].axis('off')

plt.suptitle('Classification with Circlization Results', y=1.05)
plt.tight_layout()
plt.show()
```