2021

A Comparative Study of Image Segmentation Methods

TEAM 9

KHALID ALKHALDI

LATHA NARAYANI BALAJI

NGOC PHAN

PHILLIP MERRITT

VAMSHI KANDALA

Project Name, Participants, and Workflow

Project Name

A Comparative Study of Image Segmentation Methods

Participants

Name	Email
Latha Narayani Balaji	lathanarayanibalaji@my.unt.edu
Ngoc Phan	ngocphan@my.unt.edu
Phillip Merritt	phillipmerritt@my.unt.edu
Vamshi Kandala	vamshikandala@my.unt.edu
Khalid Alkhaldi	khalidalkhaldi@my.unt.edu

Workflow

We use the followings for file storages and code version control:

- Google Drive was used to share large files.
- GitHub was used as the code repository.

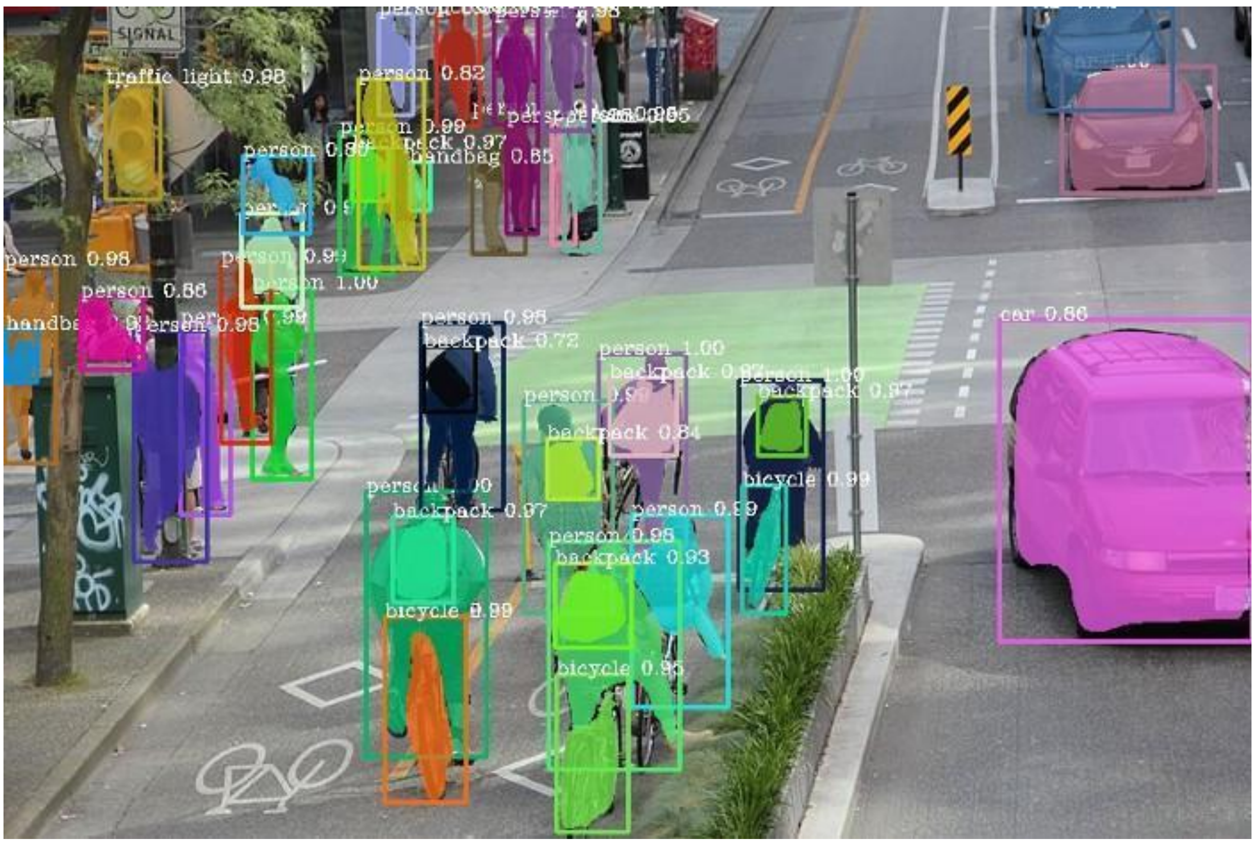
The table below shows the task assignment for each team member.

Task	Team Member
Data Specification & Pre-processing	Ngoc Phan
Model Development: U-Net Model	Ngoc Phan
Image Segmentation - Threshold Method	Vamshi Kandala
Image Segmentation - Otsu's Method	Khalid Alkhaldi
Image Segmentation - Edge Based	Latha Narayani Balaji

Image Segmentation - Region Adjacency	Phillip Merritt
Evaluation and Comparison	Phillip Merritt

Project Abstract

If you have ever used photo editing software or an app to remove the background from an image you have performed what is called Image Segmentation. More specifically, image segmentation is the task of extracting one or more objects from an image.



An example of image segmentation and object classification. [Source]

This is a classic computer vision problem with a variety of old and new methods to accomplish it. In this project we will test and compare a handful of these methods by performing segmentation on a dataset with ground truth segmentations to compare to.

Data Specification

The Oxford-IIIT pet dataset contains 7,393 pet images. It is a 37 category pet image dataset with roughly 200 images for each class. The images have large variations in scale, pose and lighting. All images have an associated ground truth annotation of breed. The dataset can be obtained in two ways:

1) Download the following GZ files on the website located at https://www.robots.ox.ac.uk/~vgg/data/pets/

    - Dataset | https://www.robots.ox.ac.uk/~vgg/data/pets/data/images.tar.gz

      File images.tar.gz contains pet images in JPG format.

    - Ground truth data | https://www.robots.ox.ac.uk/~vgg/data/pets/data/annotations.tar.gz

      File annotations.tar.gz contains annotated ground truth images in PNG format for the corresponding pet images. The annotated ground truth images would be used as the test targets for evaluating the results of various image segmentation methods.

    Below is an example of a pet image and the corresponding annotated ground truth.

Original Image	**Annotated Ground Truth**
	

2) Load the Oxford-IIIT pet dataset from *tensorflow_datasets* using the following code:

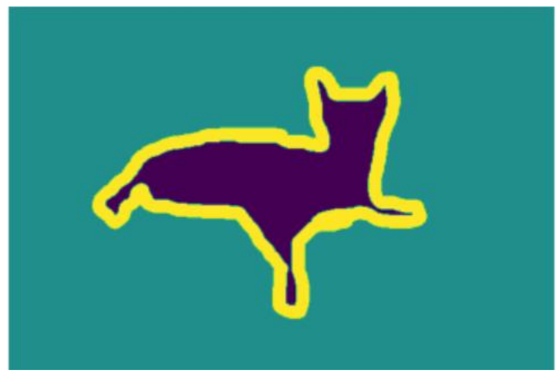
```
import tensorflow_datasets as tfds

dataset, info = tfds.load('oxford_iiit_pet:3.*.*',
```

```
with_info=True)
```

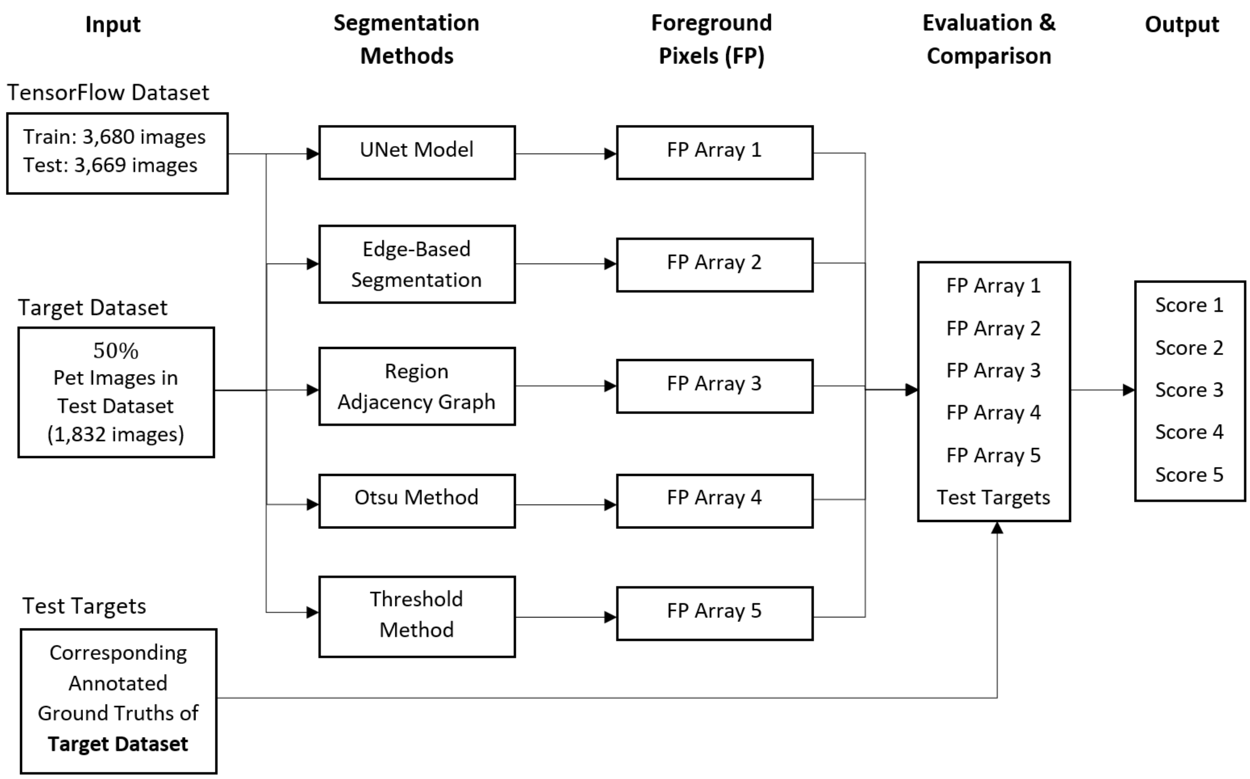
The tensorflow dataset has separated train and test sets. Train dataset has 3,680 images, and the test dataset has 3,669 images. Both train and test datasets contain the original pet images and the corresponding true mask images which are used for model development.

Below is an example of a pet image and the corresponding true mask.

Original Image	**True Mask**
	

Project Design

The figure below shows the process flow diagram for the project. First, we train an image segmentation model using the separated train and test datasets obtained from *tensorflow_datasets*. In the meantime, we randomly select 50% pet images (1,832 images) from the test dataset and the corresponding annotated ground truths for evaluation and comparison. Then, we apply five different segmentation methods to produce segmented images for the selected images and obtain the foreground pixels for each segmented image and output the results as an array. The foreground pixels arrays are then used for evaluation and cross comparison. The output score for each segmentation method is obtained by evaluating the foreground pixels array against the annotated ground truth array which is also called the test targets.

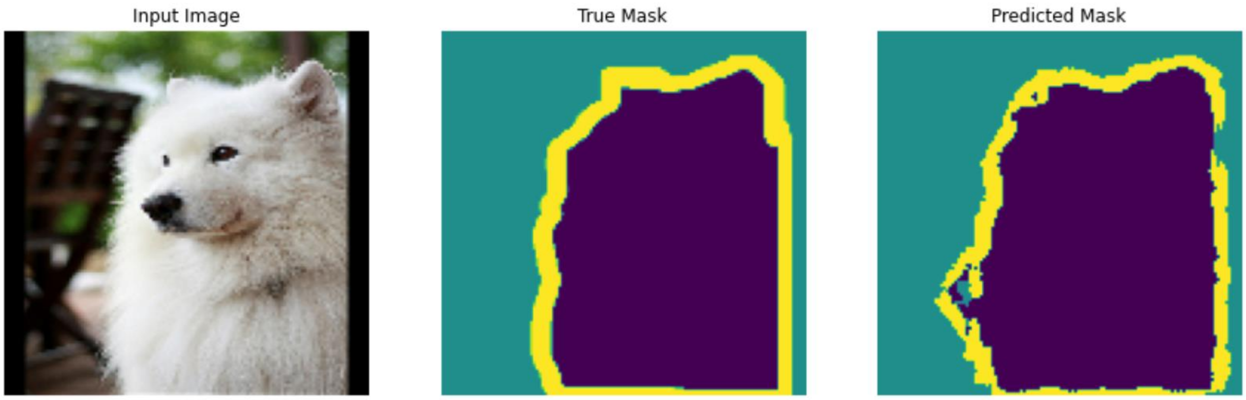


Technologies

- Google Colab for training an image segmentation model.
- Jupyter Notebook for writing Python code and presenting project demonstration.
- Programming language: Python
- Python modules
  - File operations: os, zipfile, pickle
  - Model training, image processing and visualization: TensorFlow, keras, sklearn, matplotlib, PIL, NumPy, OpenCV, numpy
  - Interactive HTML widgets for Jupyter notebooks: ipywidgets.

Image Segmentation Model

The goal of an image segmentation model is to train a Neural Network which can return a pixel-wise mask of the image. An online tutorial has been used for developing the model and can also be found at https://thecleverprogrammer.com/2020/07/22/image-segmentation/. The model is developed using the pet dataset loaded from *tensorflow_datasets*. The screenshot below shows the result of the model's prediction.



The following steps have been performed for model development.

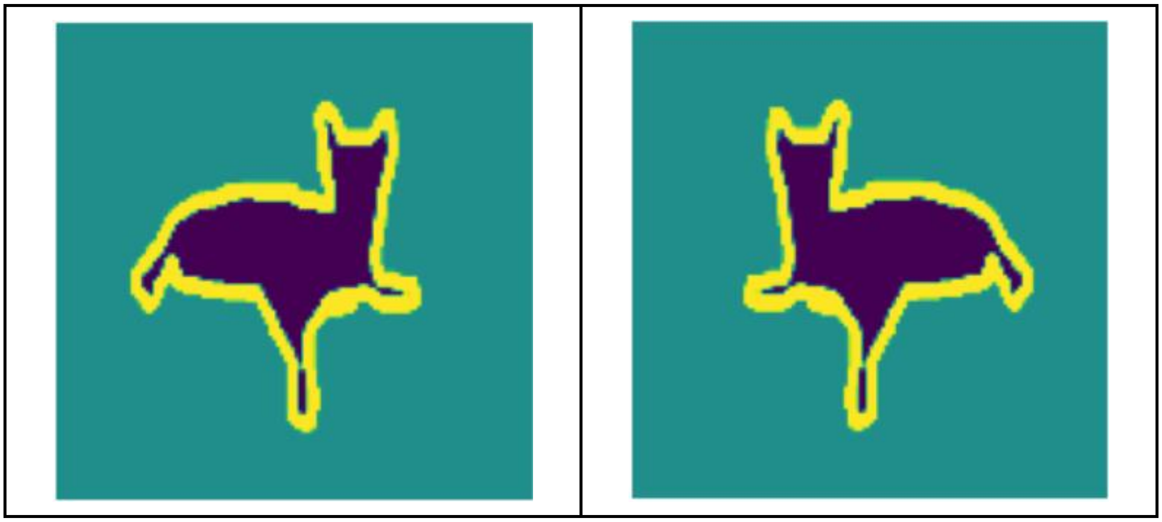
- **Data Preprocessing**

Before training the model, we performed data preprocessing on both original and true mask images. The following image preprocessing steps have been done:

- Resize the image to a size of 128 x 128.
- Perform a simple image augmentation by dividing the input image by 255.0 and subtracting 1 from the true mask.
- Randomly select images in the training dataset for flipping horizontally from left to right.

Below is an example of a pet image after preprocessing has been done.

Preprocessed Image	**Preprocessed Image** **(with Horizontal Flip)**
	



- **Model Development: U-Net Model**

  U-Net Model contains an encoder and a decoder. In order to learn the robust features and reduce all the trainable parameters, a pretrained model, used as the encoder, is loaded from *tf.keras.applications*. The model has been trained on Google Colab. Twenty epochs were used for model training, and the model has been compiled using Adam optimizer, Sparse Categorical Cross Entropy loss, and accuracy metric. Model's validation accuracy is 93.9%. The loss values for training and validation is shown in the figure below.

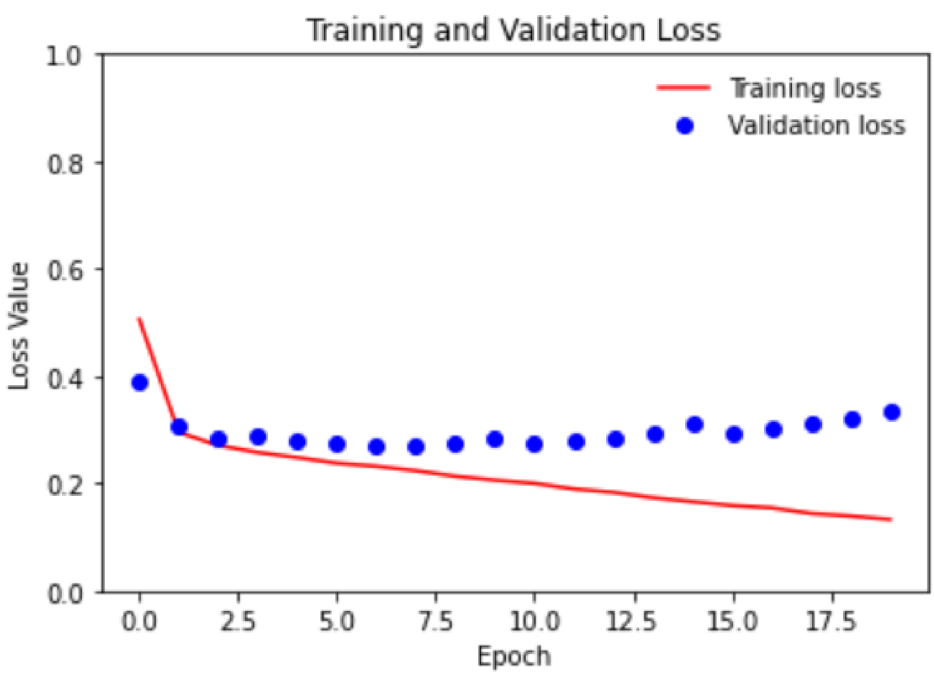


Image Segmentation Methods

1. **Threshold Method**

It is the simplest and yet powerful method for image segmentation. It is used for partitioning images directly into regions based on intensity values or properties of these values. One obvious way to extract the objects from the background is to select the value called a threshold. If the image pixel is greater than the threshold value, we call this pixel as object point, otherwise, the pixel is the background point. In addition, there are different types of threshold methods, for instance, global threshold and variable threshold. In the Global threshold method, the threshold value is constant for whole image pixels. For every pixel, the same threshold value is applied. If the pixel value is smaller than the threshold, it is set to 0, otherwise it is set to a maximum value. The function *cv.threshold* is used to apply the thresholding. The first argument is the source image, which should be a grayscale image. The second argument is the threshold value which is used to classify the pixel values. The third argument is the maximum value which is assigned to pixel values exceeding the threshold.

---

ret,th = cv.threshold(first argument, second argument, third argument,cv2.THRESH_BINARY)

Most frequently, we use thresholding to select areas of interest of an image, while ignoring the parts we are not concerned with. Global Thresholding is one of the simplest methods for image segmentation and has least computation cost. The threshold method uses local property as well grey level information for threshold selection and works well even in noisy images. In this method, we use one global value as a threshold. However, this might not be good in all cases, e.g., if an image has different lighting conditions in different areas. In that case, adaptive thresholding can help.

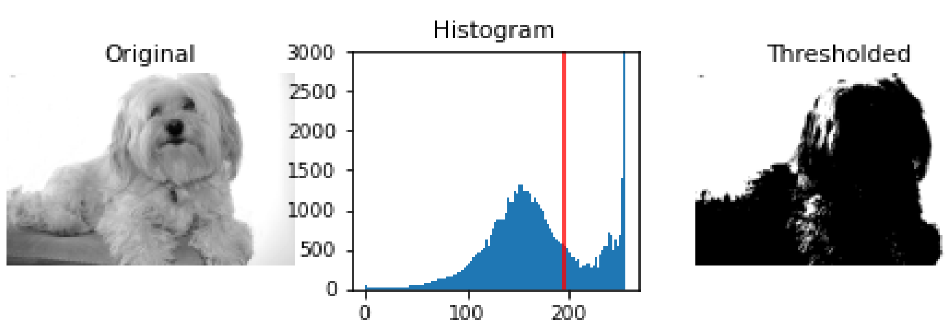
The figure below shows the result of segmentation by thresholding. The original image contains a black-haired dog on a considerably bright background. Pixel intensities vary between 0 and 255. The threshold, T = 127, was selected as the minimum between two modes on a histogram segmented image and the result of segmentation shown in the below figure, where pixels with intensity values higher than 127 are shown in white, which was applied to the segmented image.

Original Image	Simple/Global Threshold segmentation
	

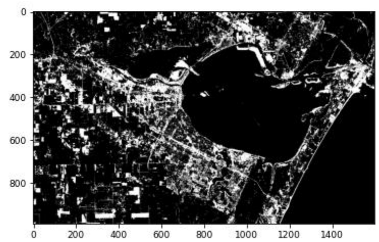
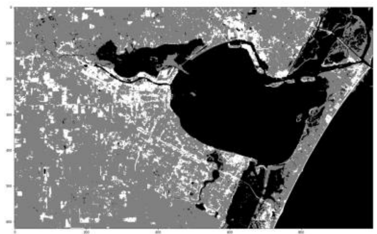
2. Otsu Method

The global threshold may work well in some images, but it does not in some others because it fails to segment the object from the background because the intensity values could be closed between the object point and background point. So, in many cases, we need to know how to determine the proper value for thresholding a specific image. Otsu’s method uses the important property which is based on the histogram of an image. we can approximately choose a value in the middle of the peaks that we get from the histogram of an image.

The basic idea behind the Otsu threshold is to split an image histogram into two classes based on the weighted variance of these groups. Based on Otsu's paper[4], one way to find the value of the threshold is to increase the variance between these classes. By using the standard method of Otsu threshold, there is a limitation if the goal is to segment the image based on multi-threshold values. However, we could extend the Otsu method to work with multi-threshold values and get the proper values by taking more than two classes that appear as a peak in the histogram of the image.

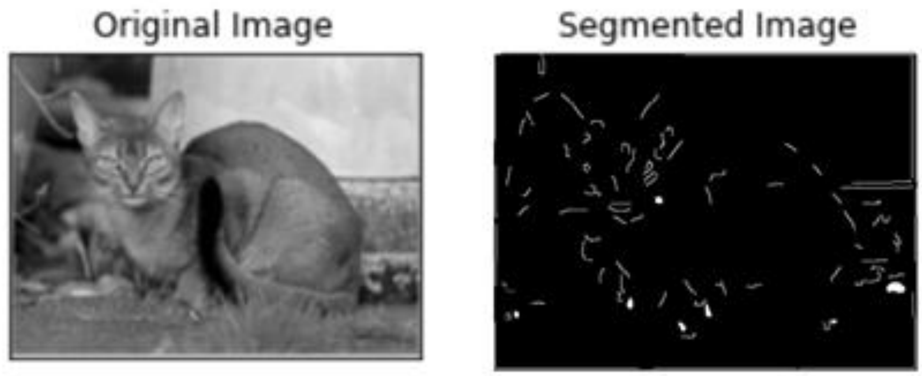


Many images have several objects that could be segmented, and by using only one threshold value, we potentially lose the majority of these objects' features in an image therefore we are unable to segment them all. The figure below shows the differences between using one threshold value and multi-values of the Otsu method.

RGB original image	One-value Otsu result	Multi- values Otsu result
		

3. Edge-Based Segmentation Method

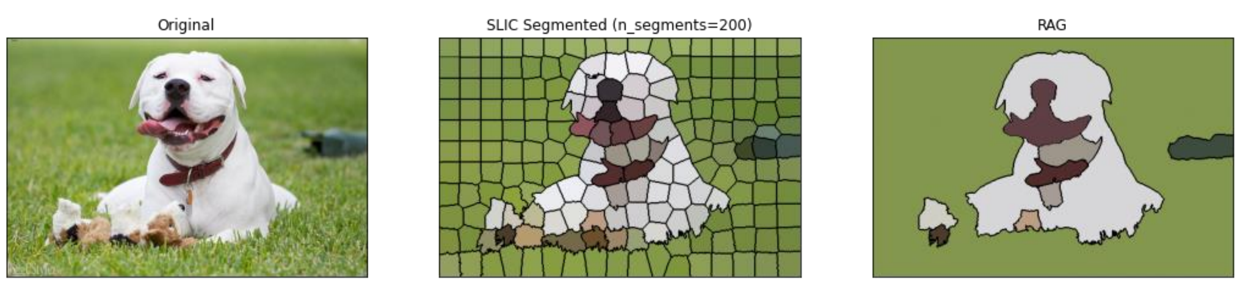
Edge Based detection methods are useful to understand image features. It shows a lot of image information that is important when we try to detect the object. Also, it reduces the size of the image which helps to increase the performance of the operations that would be applied during the image segmentation process. Edge Based segmentation method detects the edges based on many discontinuities in the grayscale image.To do this, the first step is to get the edges of features using the Canny edge detector, this algorithm is composed of five steps. Noise reduction, Gradient calculation, Non-maximum suppression, Double threshold, Edge Tracking by Hysteresis and algorithm is based on grayscale pictures. Therefore, the prerequisite is to convert the image to grayscale before applying the above-mentioned steps. After applying these steps. the following is obtained.



4. **Region Adjacency Graph (RAG) Merging**

RAG segmentation is a two part process. First, the image is segmented using the popular SLIC (Simple Linear Iterative Clustering) algorithm. It works by using k-mean clustering to combine pixels based on their proximity to one another and how similar their colors are. When using SLIC, the user can specify how many segments to cluster the pixels into and the relative importance of proximity and color similarity when deciding which pixels to cluster. The output of SLIC generally has a tiled look to it and this is far too many segments for normal segmentation tasks. This is where RAG merging comes into play.

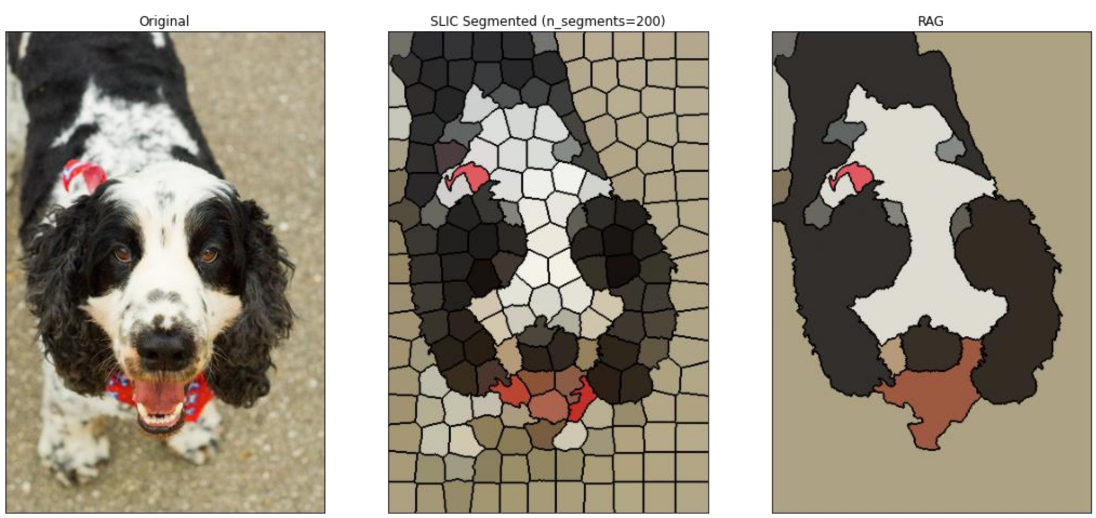
RAG merging works by combining segments based on the same metrics of similarity as before. The primary parameter to be tuned here is the threshold that determines if two segments are similar or not. Segments are repeatedly merged until there are no remaining similar segments. [5]



SLIC segmentation and RAG merging.

RAG segmentation method performs the best when there is a distinct color contrast between the subject and the background. We can see in the following image how the details of the cat are lost during merging because of the relative similarity of the lighter fur and the background compared to the darker fur.

Another issue arises when the subject is composed of a variety of colors. We can end up with a cleanly segmented background, but a variety of segments for the subject. There is no clear way of determining which of the segmentations to keep.



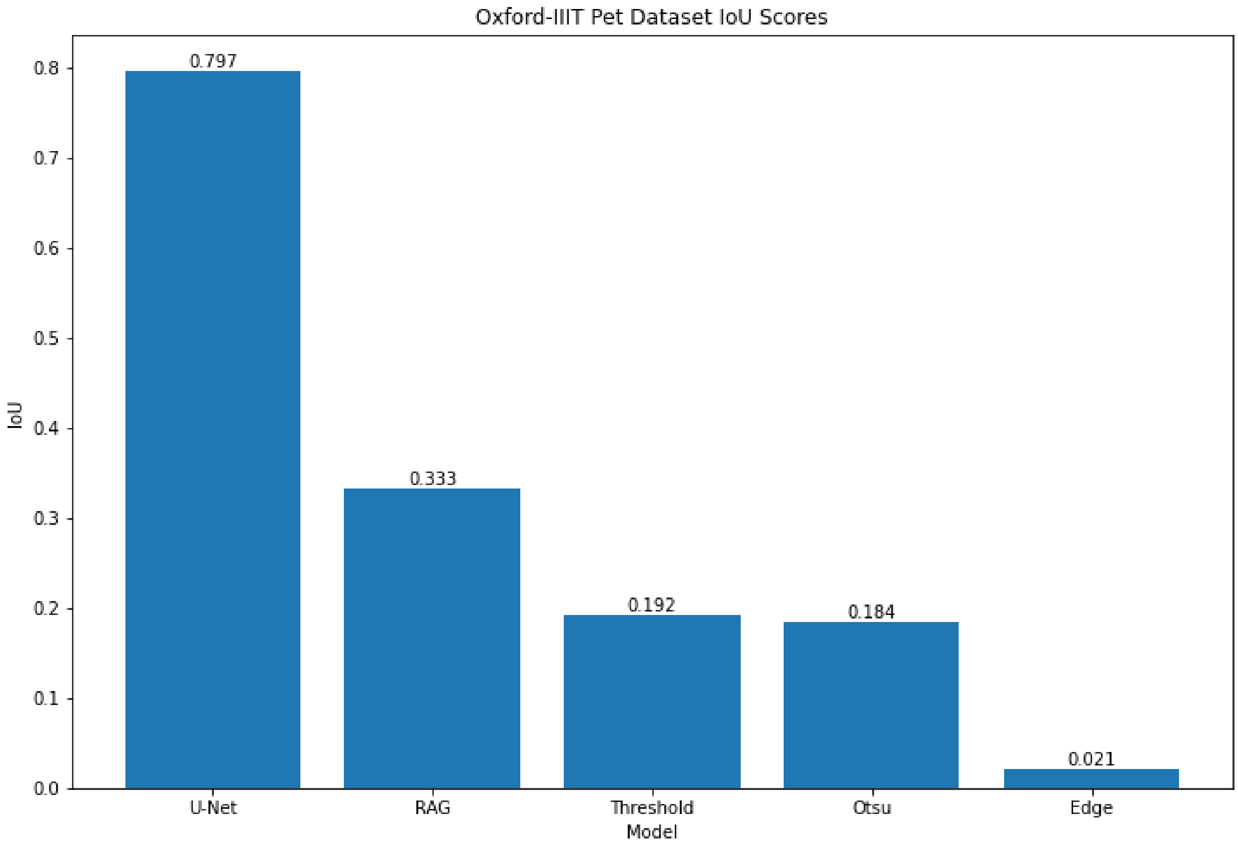
An illustration of RAG merging failing to cleanly segment a subject composed of several colors.

Evaluation & Comparison

The models were tested on a randomly sampled subset of The Oxford-IIIT Pet Dataset test set. The coordinates of the segmented foreground pixels of our models were compared against the ground truth foreground pixels. The comparison was performed by calculating IoU (Intersection over Union) scores given by the equation:

$$IoU(a, b) = a \cap b / a \cup b$$

Another way to look at it is that it is the area that the segmentations overlap divided by the area that they cover combined. It is essentially just accuracy yet it is one of the most common metrics for evaluating image segmentations against a ground truth. [6]



As you can see from the results, the U-Net model performed the best while the RAG method did a decent job. The RAG method struggled with multi-colored animals and the breeds that blended in with the background more. The two thresholding methods performed similarly. The issue with thresholding methods and this type of use case is the noisy backgrounds and complex subjects. This noise leads to a lot of extra pixels being included in the segmentation and a lot of foreground pixels being mistaken for background pixels. The edge based segmentation method performed horribly on this dataset due to being able to find clean edges to fill into objects. This mostly due to the irregular and sometimes fuzzy shape of the subject matter.

It shouldn’t come as much of a surprise that the deep neural network trained on this dataset performed the best. This does not mean that the other models are as bad as they might look here. If you only need to segment simple objects that are clearly defined against the background, you can likely avoid taking the time to train a deep model and just tune one of the classic methods for your use case. However, when you need to segment complex objects in a variety of lighting conditions and you have the data to train a model you will generally be best served by training your deep model.

Future Improvement

Currently we are using an interactive Jupyter Notebook to demonstrate the results of our project. However, everytime we demonstrate the project, we need to re-run the whole notebook. To address the issue, we could build a Flask web application to showcase the project. Another potential improvement is saving part of the test dataset loaded from *tensorflow_datasets* for model's evaluation. The Python program for generating the Oxford III Pet Dataset loaded from tensorflow_datasets can be found at https://github.com/tensorflow/datasets/blob/master/tensorflow_datasets/image_classification/oxford_iiit_pet.py. After downloading the Python file, we could modify the program to generate three separated datasets: train, validation, and test. Finally, we could build a confusion matrix to evaluate the model's performance based on the new test dataset. The online tutorial that uses a confusion matrix to evaluate the performance of an image segmentation model can be accessed at https://www.kite.com/blog/python/image-segmentation-tutorial/.

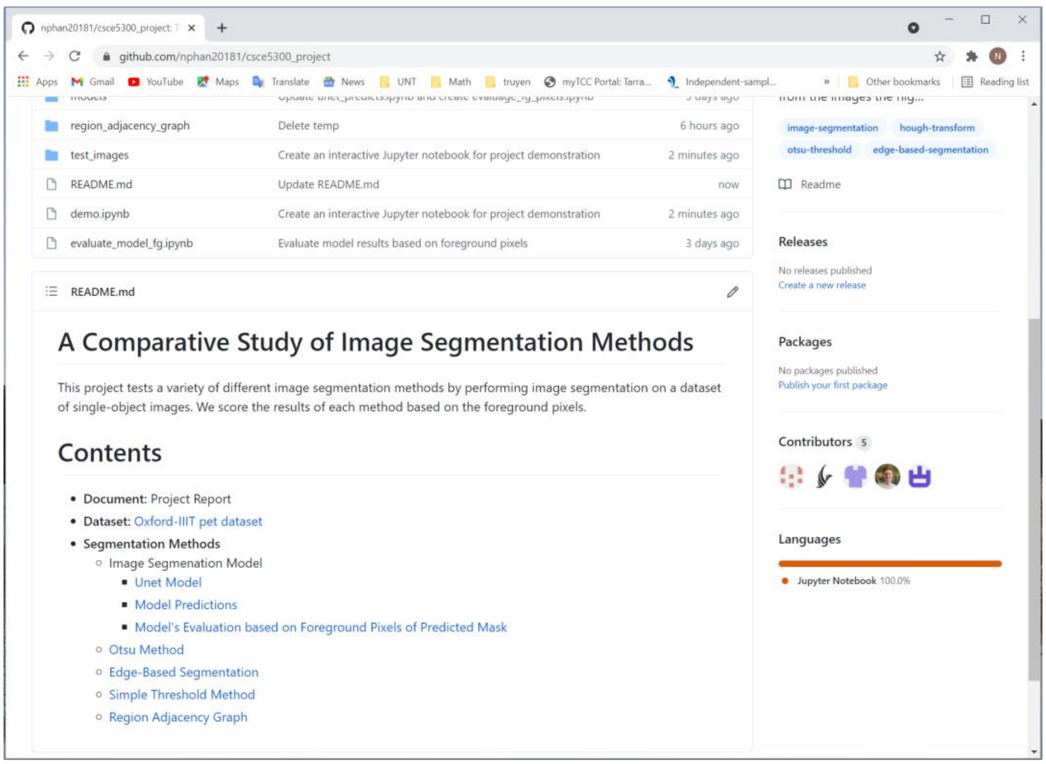
Milestones

Project Start date **Project End date**	**02 March 2021** **06 April 2021**	
Milestones	**Start**	**End**
Project Brainstorming	02 March 2021	07 March 2021
Project Proposal & Requirements gathering • Research tutorial on image segmentation model • Research image segmentation techniques • Collect dataset for image segmentation • Work on Project Proposal	02 March 2021	09 March 2021
Implementation • Train an image segmentation model	10 March 2021 15 March 2021	14 March 2021 22 March 2021

• Produce segmented images on the original images using various image segmentation techniques • Produce foreground pixels for segmented images	23 March 2021	29 March 2021
Evaluation & Comparison	30 March 2021	02 April 2021
Project Report & PowerPoint Presentation	03 April 2021	06 April 2021

GitHub Repository

- Url: https://github.com/nphan20181/csce5300_project.
- Screenshot of the repository:



Appendix

demo.ipynb

- An interactive Jupyter notebook for project demonstration.
- Url: https://github.com/nphan20181/csce5300_project/blob/main/demo.ipynb

```
from pathlib import Path
import tensorflow as tf
```

```
# get a list of jpg files in directory 'test_images'
img_folder = Path('test_images').rglob('*.jpg')
files = [x for x in img_folder]

# load saved model
model = tf.keras.models.load_model('models/unet_model.h5')
```

```
def create_mask(input_image):
    '''Produce predicted mask for input image.'''

    # get current image's size
    w, h, _ = input_image.shape

    # prepare image for model's prediction
    image_m = tf.image.resize(input_image, (128, 128))     # resize image
    image_m = tf.cast(image_m, tf.float32) / 255.0         # normalize image
    image_m = image_m[None,:,:]

    # get predicted mask that has highest score
    pred_mask = tf.argmax(model.predict(image_m), axis=-1)
    pred_mask = pred_mask[..., tf.newaxis]

    # resize image back to original size
    img_mask = tf.image.resize(pred_mask[0], (w, h))

    return img_mask
```

```
from skimage import segmentation
from skimage.future import graph

def _weight_mean_color(graph, src, dst, n):
  diff = graph.nodes[dst]['mean color'] - graph.nodes[n]['mean color']
  diff = np.linalg.norm(diff)
  return {'weight': diff}

def merge_mean_color(graph, src, dst):
  graph.nodes[dst]['total color'] += graph.nodes[src]['total color']
  graph.nodes[dst]['pixel count'] += graph.nodes[src]['pixel count']
  graph.nodes[dst]['mean color'] = (graph.nodes[dst]['total color'] /
   graph.nodes[dst]['pixel count'])
```

```
# RAG Method
def segment_image(img, compactness=60, thresh=80, n_segments=200):
```

```
    labels = segmentation.slic(img, compactness=compactness, n_segments=n_segments,
start_label=1)
    g = graph.rag_mean_color(img, labels)
    labels2 = graph.merge_hierarchical(labels, g, thresh=thresh, rag_copy=False,
                      in_place_merge=True, merge_func=merge_mean_color,
                      weight_func=_weight_mean_color)

    return labels
```

```
from skimage.io import imread
import cv2
import numpy as np

methods = ['Input Image', 'Unet Model', 'Edge-Based Method', 'Otsu Method',
           'Threshold Method', 'Region Adjacency Graph']
images = []       # store a list of input image and the corresponding segmented
images
names = []        # a list of original image names
indexes = []

# produce segmented images for all input images
for i, img in enumerate(files):
    temp = []  # store images for each pet
    names.append(str(img).split('\\')[-1].split('.')[0])   # get image's name

    # save input image to a list
    image = np.array(imread(img))
    temp.append(image)

    # get predicted mask for input image
    temp.append(create_mask(image))

    # image processing for Otsu & Edge-based methods
    image_otsu = cv2.imread(str(img), cv2.IMREAD_COLOR)
    image_otsu = cv2.cvtColor(image_otsu, cv2.COLOR_BGR2GRAY)
    scale_percent = 50
    width = int(image_otsu.shape[1] * scale_percent / 100)
    height = int(image_otsu.shape[0] * scale_percent / 100)
    dim = (width, height)
    image_otsu = cv2.GaussianBlur(image_otsu, (3, 3), 0)
    image_otsu = cv2.resize(image_otsu, dim)

    # edge-based method
    temp.append(cv2.Canny(image_otsu,100,200))

    # Otsu method
```

```
    otsu_threshold, otsout = cv2.threshold(image_otsu, 0, 255,
                                            cv2.THRESH_BINARY + cv2.THRESH_OTSU)
     otsout = cv2.normalize(otsout, None, alpha=0, beta=1,
norm_type=cv2.NORM_MINMAX)
    temp.append(otsout)

    # Threshold method
    _, th = cv2.threshold(image, 127, 255, cv2.THRESH_BINARY)
    temp.append(th)

    # RAG method
    temp.append(segment_image(image))

     # add input image and segmented images to the images list
    images.append(temp)
    indexes.append(i)
```

```
import matplotlib.pyplot as plt
from skimage import color

def show_images(menu_value, checkbox_value):
    # get image's index
    index = names.index(menu_value)

    # set figure's size
    _ = plt.figure(figsize=(12, 10))

    # show input images and all segmented images
    for i, title in enumerate(methods):
        # set image's position
        plt.subplot(2, 3, i+1)

        # plot image
        if checkbox_value:
            if title == 'Threshold Method':
                image = color.rgb2gray(images[index][i])
                plt.imshow(image, 'gray')
            else:
                plt.imshow(images[index][i], 'gray')
        else:
            plt.imshow(images[index][i])

        plt.axis('off')                 # turn off axis
          plt.title(title, fontsize=16, color='blue')   # set title

    # show plot
```

```
    plt.show()
```

```
def on_button_clicked(_):
    # "linking function with output"
    with out:
        # what happens when we press the button
        clear_output()
        if len(menu.value) > 0:
            show_images(menu.value, checkbox.value)
```

```
# some handy functions to use along widgets
from IPython.display import display, Markdown, clear_output
import ipywidgets as widgets

# build menu select box for selecting image
menu = widgets.Dropdown(options = [''] + names, values = indexes,
description='Select image:')

checkbox = widgets.Checkbox(value=False, description='Grayscale', disabled=False,)

# build submit button
button = widgets.Button(description='Perform Segmenation')

# placeholder for displaying images
out = widgets.Output()

# linking button and function together using a button's method
button.on_click(on_button_clicked)
```

```
# show image select box and submit button
widgets.VBox([menu, checkbox, button, out])
```

unet_model.ipynb

- Train image segmentation model, Unet.
- Url: https://github.com/nphan20181/csce5300_project/blob/main/models/unet_model.ipynb

```
!pip install -q git+https://github.com/tensorflow/examples.git
import tensorflow as tf
from tensorflow_examples.models.pix2pix import pix2pix

import tensorflow_datasets as tfds
tfds.disable_progress_bar()
```

```
from IPython.display import clear_output
import matplotlib.pyplot as plt
```

```
# use the Oxford-IIIT Pets dataset, that is already included in Tensorflow:
dataset, info = tfds.load('oxford_iiit_pet:3.*.*', with_info=True)
```

```
def normalize(input_image, input_mask):
  input_image = tf.cast(input_image, tf.float32) / 255.0
  input_mask -= 1
  return input_image, input_mask

@tf.function
def load_image_train(datapoint):
  input_image = tf.image.resize(datapoint['image'], (128, 128))
  input_mask = tf.image.resize(datapoint['segmentation_mask'], (128, 128))

  if tf.random.uniform(()) > 0.5:
    input_image = tf.image.flip_left_right(input_image)
    input_mask = tf.image.flip_left_right(input_mask)

  input_image, input_mask = normalize(input_image, input_mask)

  return input_image, input_mask

def load_image_test(datapoint):
  input_image = tf.image.resize(datapoint['image'], (128, 128))
  input_mask = tf.image.resize(datapoint['segmentation_mask'], (128, 128))

  input_image, input_mask = normalize(input_image, input_mask)

  return input_image, input_mask
```

```
TRAIN_LENGTH = info.splits['train'].num_examples
BATCH_SIZE = 64
BUFFER_SIZE = 1000
STEPS_PER_EPOCH = TRAIN_LENGTH // BATCH_SIZE

train = dataset['train'].map(load_image_train,
num_parallel_calls=tf.data.experimental.AUTOTUNE)
test = dataset['test'].map(load_image_test)

train_dataset = train.cache().shuffle(BUFFER_SIZE).batch(BATCH_SIZE).repeat()
train_dataset = train_dataset.prefetch(buffer_size=tf.data.experimental.AUTOTUNE)
```

```
test_dataset = test.batch(BATCH_SIZE)
```

```
# Now let's have a quick look at an image and it's mask from the data
def display(display_list):
  plt.figure(figsize=(15, 15))

  title = ['Input Image', 'True Mask', 'Predicted Mask']

  for i in range(len(display_list)):
    plt.subplot(1, len(display_list), i+1)
    plt.title(title[i])
    plt.imshow(tf.keras.preprocessing.image.array_to_img(display_list[i]))
    plt.axis('off')
  plt.show()

for image, mask in train.take(1):
  sample_image, sample_mask = image, mask
display([sample_image, sample_mask])
```

```
OUTPUT_CHANNELS = 3

base_model = tf.keras.applications.MobileNetV2(input_shape=[128, 128, 3],
include_top=False)

# Use the activations of these layers
layer_names = [
    'block_1_expand_relu',   # 64x64
    'block_3_expand_relu',   # 32x32
    'block_6_expand_relu',   # 16x16
    'block_13_expand_relu',  # 8x8
    'block_16_project',      # 4x4
]
layers = [base_model.get_layer(name).output for name in layer_names]

# Create the feature extraction model
down_stack = tf.keras.Model(inputs=base_model.input, outputs=layers)

down_stack.trainable = False
```

```
up_stack = [
    pix2pix.upsample(512, 3),  # 4x4 -&gt; 8x8
    pix2pix.upsample(256, 3),  # 8x8 -&gt; 16x16
    pix2pix.upsample(128, 3),  # 16x16 -&gt; 32x32
    pix2pix.upsample(64, 3),   # 32x32 -&gt; 64x64
```

```
]

def unet_model(output_channels):
  inputs = tf.keras.layers.Input(shape=[128, 128, 3])
  x = inputs

  # Downsampling through the model
  skips = down_stack(x)
  x = skips[-1]
  skips = reversed(skips[:-1])

  # Upsampling and establishing the skip connections
  for up, skip in zip(up_stack, skips):
    x = up(x)
    concat = tf.keras.layers.Concatenate()
    x = concat([x, skip])

  # This is the last layer of the model
  last = tf.keras.layers.Conv2DTranspose(
      output_channels, 3, strides=2,
      padding='same')  #64x64 -&gt; 128x128

  x = last(x)

  return tf.keras.Model(inputs=inputs, outputs=x)
```

```
model = unet_model(OUTPUT_CHANNELS)
model.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])
```

```
tf.keras.utils.plot_model(model, show_shapes=True)
```

```
def create_mask(pred_mask):
  pred_mask = tf.argmax(pred_mask, axis=-1)
  pred_mask = pred_mask[..., tf.newaxis]
  return pred_mask[0]

def show_predictions(dataset=None, num=1):
  if dataset:
    for image, mask in dataset.take(num):
      pred_mask = model.predict(image)
      display([image[0], mask[0], create_mask(pred_mask)])
  else:
```

```
    display([sample_image, sample_mask,
             create_mask(model.predict(sample_image[tf.newaxis, ...]))])
show_predictions()
```

```
class DisplayCallback(tf.keras.callbacks.Callback):
  def on_epoch_end(self, epoch, logs=None):
    clear_output(wait=True)
    show_predictions()
    print ('\nSample Prediction after epoch {}\n'.format(epoch+1))

EPOCHS = 20
VAL_SUBSPLITS = 5
VALIDATION_STEPS = info.splits['test'].num_examples//BATCH_SIZE//VAL_SUBSPLITS

model_history = model.fit(train_dataset, epochs=EPOCHS,
                          steps_per_epoch=STEPS_PER_EPOCH,
                          validation_steps=VALIDATION_STEPS,
                          validation_data=test_dataset,
                          callbacks=[DisplayCallback()])
```

```
loss = model_history.history['loss']
val_loss = model_history.history['val_loss']

epochs = range(EPOCHS)

plt.figure()
plt.plot(epochs, loss, 'r', label='Training loss')
plt.plot(epochs, val_loss, 'bo', label='Validation loss')
plt.title('Training and Validation Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss Value')
plt.ylim([0, 1])
plt.legend(frameon=False)
plt.show()
```

```
import numpy as np

# model's accuracy
model_history.history['accuracy'][np.argmin(model_history.history['loss'])]
```

```
show_predictions(test_dataset, 3)
```

```
model.save('drive/MyDrive/app/unet_model.h5')
```

unet_predicts.ipynb

- Unet Model Prediction
- Url: https://github.com/nphan20181/csce5300_project/blob/main/models/unet_predicts.ipynb

```
import tensorflow as tf
import numpy as np

# load saved model
model = tf.keras.models.load_model('unet_model.h5')
```

```
def create_mask(input_image):
  # get current image's size
  w, h, _ = input_image.shape

  # prepare image for model's prediction
  input_image = tf.image.resize(input_image, (128, 128))     # resize image
  input_image = tf.cast(input_image, tf.float32) / 255.0     # normalize image
  input_image = input_image[None,:,:]

  # get predicted mask
  pred_mask = tf.argmax(model.predict(input_image), axis=-1)
  pred_mask = pred_mask[..., tf.newaxis]

  # resize image back to original size
  out_image = tf.image.resize(pred_mask[0], (w, h))

  return out_image
```

```
from zipfile import ZipFile
from skimage.io import imread

# read input images from a zip file and produce predicted mask
def loadData(filepath):
    predicted_masks = []   # store predicted mask images

    with ZipFile(filepath, 'r') as zipdata:
        namelist = zipdata.namelist()

        # loop through every directory/file
        for file_path in namelist:
            # read image files and save to list
            if '.jpg' in file_path or '.JPG' in file_path:
```

```
                # load image into list
                img_bytes = zipdata.open(file_path)
                image = np.array(imread(img_bytes))            # read image byte
from zip file
                predicted_masks.append(create_mask(image))     # get predicted mask
image

    # close zip file
    zipdata.close()

    # return a list of predicted mask images
    return predicted_masks
```

```
# load input images and get predicted masks
masks = loadData('../data/Oxford-IIIT Pet test.zip')
```

```
def get_foreground(segmented_image):
  np_image = segmented_image.numpy()
  foreground_pixels = []
  foreground_value = 0

  # iterate through height
  for y in range(segmented_image.shape[0]):
    # iterate through width
    for x in range(segmented_image.shape[1]):
      # if this is a foreground pixel
      if np_image[y, x] == foreground_value:
        foreground_pixels.append((x, y)) # add it to my list of (x, y) coordinate
pairs

  return foreground_pixels
```

```
fg_pixels = []      # foreground pixels of predicted mask

for predicted_mask in masks:
  fg_pixels.append(get_foreground(predicted_mask))
```

```
import pickle

# save foreground pixels
with open('../data/model_fg_pixels.pkl', 'wb') as f:
  pickle.dump(fg_pixels, f)
```

evaluate_model_fg.ipynb

- Model's evaluation based on foreground pixels of predicted mask
- Url: https://github.com/nphan20181/csce5300_project/blob/main/evaluate_model_fg.ipynb

```
import pickle

# load foreground pixels for predicted masks
with open('data/model_fg_pixels.pkl', 'rb') as f:
    fg_pixels = pickle.load(f)
```

```
import numpy as np

# load test target array
targets = np.load('data/test_targets.npy', allow_pickle=True)
```

```
from tqdm import tqdm

def iou(target_arr, prediction_arr):
    target = set([tuple(tup) for tup in target_arr])
    prediction = set([tuple(tup) for tup in prediction_arr])
    return len(target.intersection(prediction)) / len(target.union(prediction))

def score_results(truth, preds):
    total = 0.0
    for i, target in tqdm(list(enumerate(truth))):
        total += iou(target, preds[i])
    return total / len(truth)
```

```
score_results(targets, fg_pixels)
```

Otsu Method.ipynb

- Implementing the Otsu method to get the foreground pixels for the dataset.
- Url: https://github.com/nphan20181/csce5300_project/blob/main/Otsu_threshold/Otsu%20Method.ipynb

```
import matplotlib
import matplotlib.pyplot as plt
import numpy as np
import cv2 ,os
from math import ceil
import pickle
import time

def get_images_names(path):
    files_name = os.listdir( path )
    files_name.sort()
    return files_name

def read_imges(path,images_name):
    images=[]
    for image in images_name:
        img = cv2.imread(path+image,0)
        if img is not None:
            scale_percent = 50
            width = int(img.shape[1] * scale_percent / 100)
            height = int(img.shape[0] * scale_percent / 100)
            dim = (width, height)
            img = cv2.GaussianBlur(img, (3, 3), 0)
            img = cv2.resize(img, dim)
            images.append(img)
        else:
            print('Can not read image files!',path+image)
    return images

def segment_images(images):
    seg_images=[]
    for image in images:
        otsu_threshold, otsout = cv2.threshold(image, 0, 255, cv2.THRESH_BINARY + cv2.THRESH_OTSU)
        otsout = cv2.normalize(otsout, None, alpha=0, beta=1, norm_type=cv2.NORM_MINMAX)
        seg_images.append(np.array(otsout))
    return seg_images

def get_foreground(segmented_image):
    foreground_pixels = []
    foreground_value = 0
    for img in segmented_image:
        for y in range(img.shape[0]):
            for x in range(img.shape[1]):
                if img[y, x] == foreground_value:
                    foreground_pixels.append((x, y)) # add it to my list of (x, y) coordinate pairs
    return foreground_pixels

def save_fg(images):
    with open('Otsu_fg_pixels.pkl', 'wb') as f:
        pickle.dump(images, f)

def load_fg():
    with open('Otsu_fg_pixels.pkl', 'rb') as f:
        loaded_fg = pickle.load(f)
    return loaded_fg
```

```
path='test/'
#get all images name
start_time = time.time()

images_name = get_images_names(path)
images = read_imges(path,images_name)

print("--- execution time %s seconds ---" % (time.time() - start_time))
```

```
seg_img=[]
start_time = time.time()

for img in images:
    seg_img.append(segment_images(img))

print("--- execution time: %s seconds ---" % (time.time() - start_time))
```

```
fg=[]
start_time = time.time()

for img in seg_img:
    fg.append(get_foreground(img))
print("--- execution time %s seconds ---" % (time.time() - start_time))
```

```
start_time = time.time()
save_fg(fg)
print("--- execution time %s seconds ---" % (time.time() - start_time))
```

SimpleThresholding.ipynb

- Threshold image segmentation
- Url: https://github.com/nphan20181/csce5300_project/blob/main/SimpleThreshold%20Method/SimpleThresholding.ipynb

```
import cv2
import numpy as np
from matplotlib import pyplot as plt
import matplotlib.pyplot as pylab
import os
import pandas as pd
import numpy as np
from PIL import Image

data_dir = './Oxford-IIIT images/'

def read_image_files(dir):
    images = []
    filenames = os.listdir(dir)
    #print(filenames)
    for filename in filenames:
        images.append(cv2.imread(dir + filename, 0))
     return images
dataset = read_image_files(data_dir);

#Threshold.Segmentation
foreground_pixels=[]
```

```
foreground_value=0
th_data=[]
img_data=[]
def ThSeg(im):
    img=im
    img_data.append(img)
    ret,th = cv2.threshold(img,127,255,cv2.THRESH_BINARY)
    th_data.append(th)
    pixels=np.argwhere(th == foreground_value)
    foreground_pixels.append(pixels)
    return pixels



prediction=[]
for i in range(len(dataset)):
    k=ThSeg(dataset[i])
print(len(foreground_pixels))
#prediction=foreground_pixels
#prediction
np.save('SimpleThresholding5',foreground_pixels)
```

Edge_Based_Segmentation_Method.ipynb

- Edge Based Segmentation
- Url: https://github.com/nphan20181/csce5300_project/blob/main/Edge_Based_Segmentation/Edge_Based_Segmentation_Method.ipynb

```python
import matplotlib
import matplotlib.pyplot as plt
import numpy as np
import cv2 ,os
from math import ceil
import pickle
import time
from skimage import io, feature
from skimage.color import rgb2gray
from scipy import ndimage as ndi
from skimage import morphology
from skimage.feature import canny

foreground_value = 0
foreground_pixels=[]
segmented_images=[]

def read_image_files(dir):
    images = []
    filenames = os.listdir(dir)
    for filename in filenames:
        images.append(cv2.imread(dir + filename, 0))
    return images

path='./data/test/'
image_files = read_image_files(path)

def edge_based_segmentation(image):
    edges = canny(image, sigma=2)
    filled = ndi.binary_fill_holes(edges)
    filled_cleaned = morphology.remove_small_objects(filled, 21)
    segmented_images.append(filled_cleaned)
    pixels=np.argwhere(filled_cleaned == foreground_value)
    foreground_pixels.append(pixels)
    return

for img in image_files:
    edge_based_segmentation(img)

np.save('EdgeBasedSegmentation',foreground_pixels)
```

RAG_image_segmentation.ipynb

- Source code for implementing Region Adjacency Graph (RAG) method
- Url: https://github.com/nphan20181/csce5300_project/blob/main/region_adjacency_graph/RAG_image_segmentation.ipynb

[1]

```
from skimage import segmentation
from skimage.future import graph
from matplotlib import image
from skimage.color import label2rgb
import cv2
import os
import numpy as np
def _weight_mean_color(graph, src, dst, n):
  diff = graph.nodes[dst]['mean color'] - graph.nodes[n]['mean color']
  diff = np.linalg.norm(diff)
  return {'weight': diff}
def merge_mean_color(graph, src, dst):
  graph.nodes[dst]['total color'] += graph.nodes[src]['total color']
  graph.nodes[dst]['pixel count'] += graph.nodes[src]['pixel count']
  graph.nodes[dst]['mean color'] = (graph.nodes[dst]['total color'] /
  graph.nodes[dst]['pixel count'])
```

[2]

```
def segment_image(img, compactness, thresh, n_segments=200):
  labels = segmentation.slic(img, compactness=compactness, n_segments=n_segments)
  g = graph.rag_mean_color(img, labels)
  labels2 = graph.merge_hierarchical(labels, g, thresh=thresh,
    rag_copy=False,
    in_place_merge=True,
    merge_func=merge_mean_color,
    weight_func=_weight_mean_color)

  h, w = img.shape[0], img.shape[1]
  fground_label = labels2[h // 2, w // 2]

  return get_foreground(labels2, foreground_value=fground_label)

def get_foreground(segmented_image, foreground_value = 255):
  foreground_pixels = []

  indices = np.where(segmented_image == foreground_value)
  return np.stack((indices[1], indices[0]), 1)
```

```
def get_prediction(dataset): # dataset is a list of images
  prediction = []

  # for each image in the dataset
  for image in dataset:
    foreground_pixels = segment_image(image)
    # add those to your list of predictions
    prediction.append(foreground_pixels)

  return prediction
```

[8]

```
from tqdm import tqdm

comp = 60
thresh = 80
image_path = './data/test/images/'
filenames = [image_path + name for name in os.listdir(image_path)]
pred_segs = []

for path in tqdm(filenames):
  img = cv2.imread(path, 0)
  seg_img = segment_image(img, comp, thresh)
  middle_pixel_value = seg_img[seg_img.shape[0] // 2, seg_img.shape[1] // 2]
  fg = get_foreground(seg_img, foreground_value=middle_pixel_value)
  pred_segs.append(fg)
```

```
100%|██████████| 1832/1832 [00:21<00:00, 83.97it/s]
```

References

1. Kharwal, Aman. *Image Segmentation*. (March 28, 2021). Retrieved from https://thecleverprogrammer.com/2020/07/22/image-segmentation/.
2. Tensorflow. *oxford_iiit_pet*. (March 30, 2021). Retrieved from https://www.tensorflow.org/datasets/catalog/oxford_iiit_pet.
3. Jupyter Widgets. *ipywidgets*. (April 1, 2021). Retrieved from https://ipywidgets.readthedocs.io/en/latest/.
4. N. Otsu, "A Threshold Selection Method from Gray-Level Histograms," in *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 9, no. 1, pp. 62-66, Jan. 1979, doi: 10.1109/TSMC.1979.4310076.
5. Dey, Sandipan. *Hands-On Image Processing with Python*. Packt Publishing, n.d.
6. Hofesmann, Eric. "*IoU a Better Detection Evaluation Metric*." Medium, March 1, 2021. https://towardsdatascience.com/iou-a-better-detection-evaluation-metric-45a511185be1.