CSC420 A3

Ziqi Chen #1002576722

November 2, 2019

Here is the link for all the weights trained for question 1 and 2

https://drive.google.com/open?id=16Vq_8cc037gvoqVIRphpxB6TPjLENe8d

1 Image Segmentation

1.1 Implement U-NET

For the U-Net I implemented three ways to calculate loss, binary cross entropy, mean squared error and Sorensen Dice loss.

```
import numpy as np
import tensorflow as tf
import torch
import sys.float_info.epsilon as epsilon

def sorensen_dice_loss(pred, target):
    return tf.reduce_mean(tf.reduce_mean(tf.abs(pred - target)))

def mse(pred, target):
    return tf.reduce_mean(tf.reduce_mean(
        tf.math.squared_difference(pred, target)
    ))

def bce(pred, target):
    return tf.reduce_mean(
        tf.reduce_sum(
            -(target * tf.math.log(pred + epsilon) + (1 - target) * tf.math.log(1 - pred + epsilon))
        )
    )
```

Here is the structure for the UNet.

```
import numpy as np
import torch
import torch.nn as nn
from tensorflow.keras.layers import Conv2D, Conv2DTranspose, MaxPooling2D
from tensorflow.keras.layers import BatchNormalization, Activation, concatenate
from tensorflow.keras.models import Model


def conv(input, out_channels, kernel_size, batchnorm = True):
    c1 = Conv2D(filters=out_channels, kernel_size=(kernel_size, kernel_size), kernel_initializer="he_normal",padding="same")(input)
    if batchnorm:
        c1 = BatchNormalization()(c1)
    c1 = Activation("relu")(c1)
    c2 = Conv2D(filters=out_channels, kernel_size=(kernel_size, kernel_size), kernel_initializer="he_normal",padding="same")(c1)
    if batchnorm:
        c2 = BatchNormalization()(c2)
    c2 = Activation("relu")(c2)
    return c2

def UNet(input, n_channels=64, kernel_size=3, batchnorm = True):
    # Down Conv
    c1 = conv(input, n_channels, kernel_size, batchnorm)
    p1 = MaxPooling2D((2, 2))(c1)
    c2 = conv(p1, 2 * n_channels, kernel_size, batchnorm)
    p2 = MaxPooling2D((2, 2))(c2)
    c3 = conv(p2, 4 * n_channels, kernel_size, batchnorm)
    p3 = MaxPooling2D((2, 2))(c3)
    c4 = conv(p3, 8 * n_channels, kernel_size, batchnorm)
    p4 = MaxPooling2D((2, 2))(c4)
    c5 = conv(p4, 16 * n_channels, kernel_size, batchnorm)
    # Up Conv
    kernel = (2,2)
    stride = (2,2)
    u6 = Conv2DTranspose(8 * n_channels, kernel, strides=stride, padding='same')(c5)
    u6 = concatenate([u6, c4])
    c6 = conv(u6, 8 * n_channels, kernel_size, batchnorm)
    u7 = Conv2DTranspose(4 * n_channels, kernel, strides=stride, padding='same')(c6)
    u7 = concatenate([u7, c3])
    c7 = conv(u7, 4 * n_channels, kernel_size, batchnorm)
    u8 = Conv2DTranspose(2 * n_channels, kernel, strides=stride, padding='same')(c7)
    u8 = concatenate([u8, c2])
    c8 = conv(u8, 2 * n_channels, kernel_size, batchnorm)
    u9 = Conv2DTranspose(n_channels, kernel, strides=stride, padding='same')(c8)
    u9 = concatenate([u9, c1], axis=3)
    c9 = conv(u9, n_channels, kernel_size, batchnorm)

    outputs = Conv2D(1, (1, 1), activation='sigmoid')(c9)
    model = Model(inputs=[input], outputs=[outputs])
    return model
```

Code for dataset class

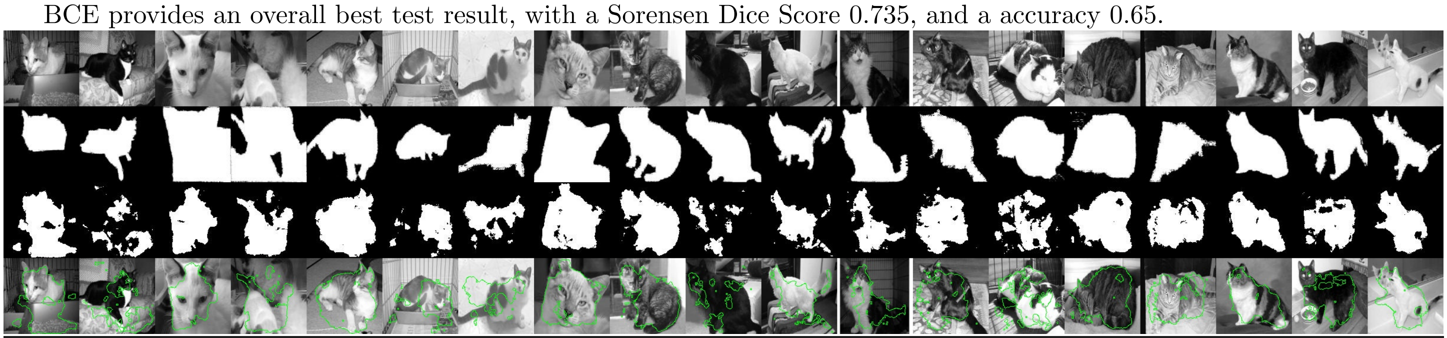
```python
import torch
import os
from torch.utils.data import Dataset, DataLoader
from tensorflow.keras.preprocessing.image import img_to_array, load_img
import tensorflow as tf
import numpy as np
from skimage.transform import resize


class CatDataset(Dataset):
    def __init__(self, rootdirectory, im_height=128, im_width=128):
        input_directory = rootdirectory + 'input/'
        mask_directory = rootdirectory + 'mask/'
        self.len = len(os.listdir(input_directory))
        self.X = np.zeros((self.len, im_height, im_width, 1), dtype=np.float32)
        self.Y = np.zeros((self.len, im_height, im_width, 1), dtype=np.float32)

        id = 0
        for inputfilename in os.listdir(input_directory):
            input_img = load_img(
                os.path.join(input_directory, inputfilename), color_mode="grayscale")
            if input_img is not None:
                input_img = img_to_array(input_img)
                input_img = resize(input_img, (int(im_height), int(im_width)))
                new_input_img = torch.from_numpy(input_img)
                self.X[id, ..., 0] = new_input_img.squeeze() / 255
            id += 1
        id = 0
        for maskfilename in os.listdir(mask_directory):
            mask_img = load_img(
                os.path.join(mask_directory, maskfilename), color_mode="grayscale")
            if mask_img is not None:
                mask_img = img_to_array(mask_img)
                mask_img = resize(mask_img, (int(im_height), int(im_width)))
                new_mask_img = torch.from_numpy(mask_img)
                self.Y[id] = new_mask_img / 255
            id += 1

    def __len__(self):
        return self.len

    def __getitem__(self, idx):
        return (self.X[idx], self.Y[idx])

    def augment(self):
        data_size = self.X.shape
        x = self.X
        y = self.Y

        # flip
        axis = np.random.randint(1, 2)
        flipped_x = np.flip(x, axis=axis)
        flipped_y = np.flip(y, axis=axis)

        # noise
        noise = np.random.randint(5, size=data_size, dtype='uint8')
        noised_x = x + noise
        noised_y = y

        # rotate
        rotated_x = np.zeros(data_size)
        rotated_y = np.zeros(data_size)

        croped_x = np.zeros(data_size)
        croped_y = np.zeros(data_size)
        for i in range(data_size[0]):
            _x = self.X[i]
            t = self.Y[i]

            k = np.random.randint(1, 3)
            rotated_x[i] = np.rot90(_x, k)
            rotated_y[i] = np.rot90(t, k)

            hd = np.random.randint(0, 1)
            wd = np.random.randint(0, 1)
            h_crop = np.random.randint(0, data_size[1]/5)
            w_crop = np.random.randint(0, data_size[2]/5)
            # crop from top if hd = 0, else from bottom
            if hd == 0:
                # crop from left if wd == 0, else from right
                if wd == 0:
                    cx = _x[h_crop:, w_crop:]
                    ct = t[h_crop:, w_crop:]
                else:
                    cx = _x[h_crop:, :-w_crop]
                    ct = t[h_crop:, :-w_crop]
            else:
                if wd == 0:
                    cx = _x[:-h_crop, w_crop:]
                    ct = t[:-h_crop, w_crop:]
                else:
                    cx = _x[:-h_crop, :-w_crop]
                    ct = t[:-h_crop, :-w_crop]
            croped_x[i] = resize(cx, (data_size[1], data_size[2]))
            croped_y[i] = resize(ct, (data_size[1], data_size[2]))
        aug_x = np.concatenate(
            (x, flipped_x, noised_x, rotated_x, croped_x), axis=0)
        aug_y = np.concatenate(
            (y, flipped_y, noised_y, rotated_y, croped_y), axis=0)

        self.X = aug_x
        self.Y = aug_y
```

Code for training model and testing model

```python
def train_model(model,out_folder, augment=False, save_path='./weights/weight.h5',
                batch_size = 10, learning_rate=0.01, momentum=0.9, loss="binary_crossentropy", path_train=path_train):
    train_dataset = dataset.CatDataset(path_train, im_height, im_width)
    if augment:
        train_dataset.augment()
    X_train, y_train = train_dataset.X, train_dataset.Y
    if loss == "sorensen":
        model.compile(optimizer=SGD(learning_rate=learning_rate,
                                    momentum=momentum), loss=loss, metrics=["accuracy", loss_functions.sorensen_dice_loss])
    else:
        model.compile(optimizer=SGD(learning_rate=learning_rate,
                                    momentum=momentum), loss=loss, metrics=["accuracy"])
    callbacks = [
        EarlyStopping(patience=10, verbose=1),
        ReduceLROnPlateau(factor=0.1, patience=3, min_lr=0.00001, verbose=1),
        ModelCheckpoint(save_path, monitor='loss', mode='min',
                        verbose=1, save_best_only=True, save_weights_only=True)
    ]
    results = model.fit(X_train, y_train, batch_size=batch_size,
                        epochs=100, callbacks=callbacks)
    return results
```

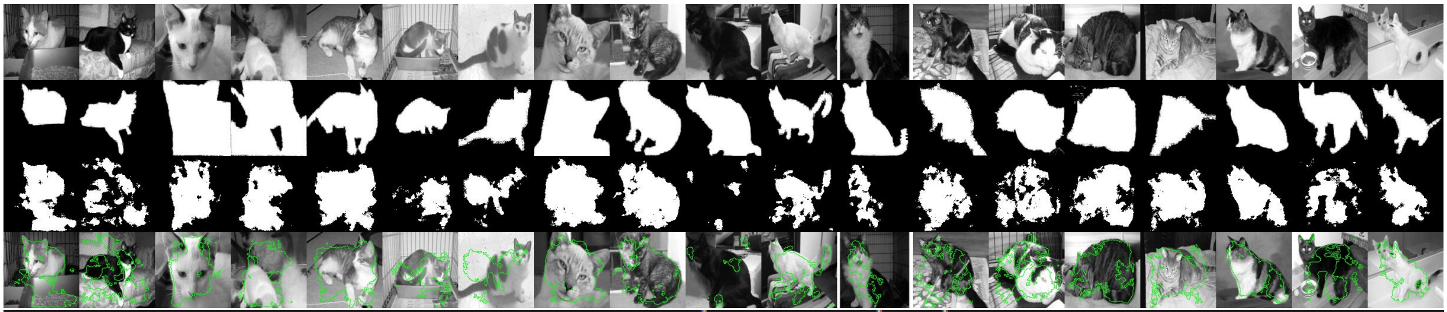
```python
def test_model(model, weight_path, save_path, threshold=0.5, path_test="./test/"):
    threshold = threshold
    test_dataset = CatDataset(path_test, 128, 128)
    X_test, y_test = test_dataset.X, test_dataset.Y
    model.load_weights(weight_path)
    pred_test = model.predict(X_test, verbose=1)

    for i in range(pred_test.shape[0]):
        test_pred = pred_test[i]
        test_y = y_test[i] * 255
        test_x = X_test[i] * 255
        test_pred[test_pred > threshold] = 255
        test_pred[test_pred <= threshold] = 0
    output = []
    for i in range(len(X_test)):
      input_img = X_test[i]
      mask_img = y_test[i]
      pred_img = pred_test[i]
      current = [input_img, mask_img, pred_img]
      mask_edge = cv.Canny(mask_img, mask_img.shape[0], mask_img.shape[1])
      pred_edge = cv.Canny(pred_img, pred_img.shape[0], pred_img.shape[1])
      where_maskedge = np.where(mask_edge==255)
      where_prededge = np.where(pred_edge==255)
      new = input_img.copy()
      new[where_prededge] = [0,255,0]
      current.append(new)
      concatenated = np.concatenate(current)
      result.append(concatenated)
    output = np.concatenate(result, axis=1)
    plt.imshow(output)
    cv.imwrite(save_path.format(inputfilename), output)
```

BCE provides an overall best test result, with a Sorensen Dice Score 0.735, and a accuracy 0.65.



```
Sorensen Dice Coefficient score is 0.7355291843414307
Sorensen Dice Coefficient loss is 0.26447081565856934
21/1 [===================================================
=========================================================
=========================================================
=========================================================
=========================================================
=========================================================
=========================================================
============================] - 1s 36ms/sample - loss: 0.7728 - accuracy
: 0.6546
```

And here is the result for mean squared error, it has a lower accuracy and lower sorensen dice coefficient.

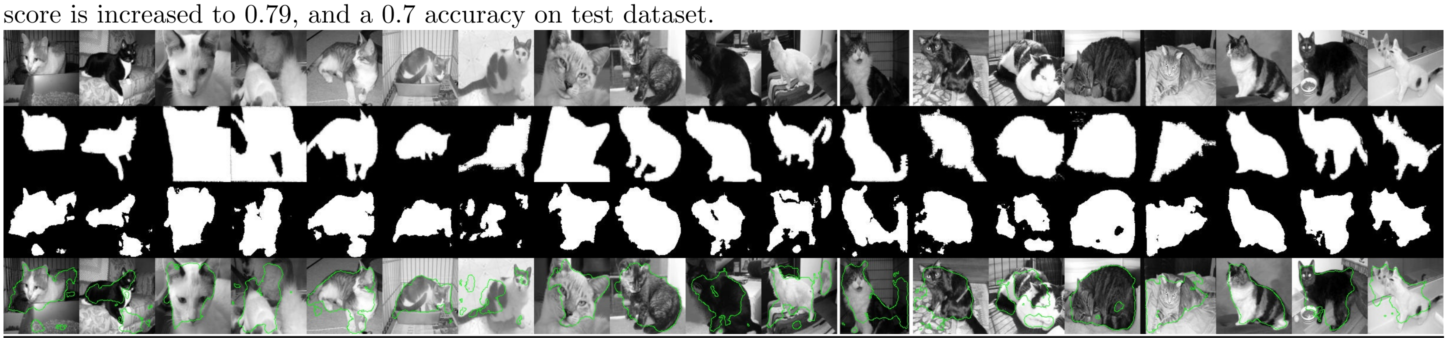


```
Sorensen Dice Coefficient result is 0.6806143522262573
Sorensen Dice Coefficient result implemented by me is 0.2756060063838959
21/1 [==========================================================================
================================================================================
================================================================================
================================================================================
================================================================================
================================================================================
================================================================================
==========================================] - 1s 28ms/sample - loss: 0.6709 - accuracy
: 0.6433
```

The reason why BCE returns a better result is because MSE penalizes correct prediction with high confidence, thus it might miss some of the correct predictions.

1.2 Data Augmentataion

I applied random flip, random chop, and random rotation on the dataset, with a total sample size of 300, the sorensen dice score is increased to 0.79, and a 0.7 accuracy on test dataset.



```
Sorensen Dice Coefficient score is 0.7927371859550476
Sorensen Dice Coefficient loss is 0.2072628289461136
```

Although the accuracy and dice score didn't increase much, but we can see in the resulted image the mask is much smoother than before and is more like the shape of a cat.
Code is shown below:

```python
def augment(self):
    data_size = self.X.shape
    x = self.X
    y = self.Y

    # flip
    axis = np.random.randint(1, 2)
    flipped_x = np.flip(x, axis=axis)
    flipped_y = np.flip(y, axis=axis)

    # noise
    noise = np.random.randint(5, size=data_size, dtype='uint8')
    noised_x = x + noise
    noised_y = y

    # rotate
    rotated_x = np.zeros(data_size)
    rotated_y = np.zeros(data_size)

    croped_x = np.zeros(data_size)
    croped_y = np.zeros(data_size)
    for i in range(data_size[0]):
        _x = self.X[i]
        t = self.Y[i]

        k = np.random.randint(1, 3)
        rotated_x[i] = np.rot90(_x, k)
        rotated_y[i] = np.rot90(t, k)

        hd = np.random.randint(0, 1)
        wd = np.random.randint(0, 1)
        h_crop = np.random.randint(0, data_size[1]/5)
        w_crop = np.random.randint(0, data_size[2]/5)
        # crop from top if hd = 0, else from bottom
        if hd == 0:
            # crop from left if wd == 0, else from right
            if wd == 0:
                cx = _x[h_crop:, w_crop:]
                ct = t[h_crop:, w_crop:]
            else:
                cx = _x[h_crop:, :-w_crop]
                ct = t[h_crop:, :-w_crop]
        else:
            if wd == 0:
                cx = _x[:-h_crop, w_crop:]
                ct = t[:-h_crop, w_crop:]
            else:
                cx = _x[:-h_crop, :-w_crop]
                ct = t[:-h_crop, :-w_crop]
        croped_x[i] = resize(cx, (data_size[1], data_size[2]))
        croped_y[i] = resize(ct, (data_size[1], data_size[2]))
    aug_x = np.concatenate(
        (x, flipped_x, noised_x, rotated_x, croped_x), axis=0)
    aug_y = np.concatenate(
        (y, flipped_y, noised_y, rotated_y, croped_y), axis=0)

    self.X = aug_x
    self.Y = aug_y
```

1.3 Transfer Learning

```python
base_model = tf.keras.applications.MobileNetV2(input_shape=img_shape,
                                               include_top=False,
                                               weights='imagenet')
base_model.trainable = False
```

```python
[5]
    model = tf.keras.Sequential([
      base_model,
      GlobalAveragePooling2D(),
      Dense(img_size * img_size, activation='relu'),
      # Dense(2 * img_size * img_size, activation='relu'),
      Dense(2 * img_size * img_size, activation='relu'),
      Dense(img_size * img_size, activation="sigmoid")
    ])
    model.summary()
```

```python
[ ]  lr= 0.001
     loss = "binary_crossentropy"
     model.compile(Adam(lr=lr),
                   loss=loss,
                   metrics=['accuracy'])
```

```
(raw_train, raw_validation, raw_test), metadata = tfds.load(
    'cats_vs_dogs', split=list(splits),
    with_info=True, as_supervised=True)
```

```
[5] model = tf.keras.Sequential([
      base_model,
      GlobalAveragePooling2D(),
      Dense(2 * img_size * img_size, activation='relu'),
      # Dense(2 * img_size * img_size, activation='relu'),
      # Dense(2 * img_size * img_size, activation='relu'),
      Dense(img_size * img_size, activation="sigmoid")
    ])
    model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
mobilenetv2_1.00_224 (Model) (None, 7, 7, 1280)        2257984
_________________________________________________________________
global_average_pooling2d (Gl (None, 1280)              0
_________________________________________________________________
dense (Dense)                (None, 100352)            128550912
_________________________________________________________________
dense_1 (Dense)              (None, 50176)             5035312128
=================================================================
Total params: 5,166,121,024
Trainable params: 5,163,863,040
Non-trainable params: 2,257,984
_________________________________________________________________
```

```
callbacks = [
       EarlyStopping(patience=10, verbose=1),
       ReduceLROnPlateau(factor=0.1, patience=3, min_lr=0.0001, verbose=1),
       ModelCheckpoint("./weight/", monitor='loss', mode='min',
                      verbose=1, save_best_only=True, save_weights_only=True)
   ]
```

```
model.fit(train_x, train_y,batch_size=30,
                      epochs=100, callbacks=callbacks)
```

1.4 Visualizing segmentation predictions

For this part I used opencv canny detection to detect the edge in mask image, for quicker implementation and better result. More details is already provided in 1.1 and 1.2, the mask for this part isn't shown.

The second segmentation failed pretty bad, unet mistakenly recognize the bed sheet as a part of the cat because their colors are too similar. 

2 Bounding Box Design

2.1 Problem definition

For this problem, the pairs for my neural network is still (input image, mask), and my loss function is either mean squared error or binary cross entropy. The reason why I'm representing it this way is because I want to use U-Net or CNN to recognize the circle as a feature of the image, and since convolutional neural nets are noise resistant, so it could work well for this problem. We can simply do a binary classify on each pixel of the image to see if it's a part of the circle. After a mask is generated we can use either opencv circle option to find a circle, or implement another neural net with one or two hidden layers and 3 outputs presenting x, y, and R. In my case I used opencv library.

2.2 Implementation

Here is the neural net structure.

```
import numpy as np
import torch
import torch.nn as nn
from tensorflow.keras.layers import Conv2D, Conv2DTranspose, MaxPooling2D
from tensorflow.keras.layers import BatchNormalization, Activation, concatenate
from tensorflow.keras.models import Model


def conv(input, out_channels, kernel_size, batchnorm = True):
    c1 = Conv2D(filters=out_channels, kernel_size=(kernel_size, kernel_size),
    kernel_initializer="he_normal",padding="same")(input)
    if batchnorm:
        c1 = BatchNormalization()(c1)
    c1 = Activation("relu")(c1)
    c2 = Conv2D(filters=out_channels, kernel_size=(kernel_size, kernel_size),
    kernel_initializer="he_normal",padding="same")(c1)
    if batchnorm:
        c2 = BatchNormalization()(c2)
    c2 = Activation("relu")(c2)
    return c2

def UNet(input, n_channels=64, kernel_size=3, batchnorm = True):
    # Down Conv
    c1 = conv(input, n_channels, kernel_size, batchnorm)
    p1 = MaxPooling2D((2, 2))(c1)
    c2 = conv(p1, 2 * n_channels, kernel_size, batchnorm)
    p2 = MaxPooling2D((2, 2))(c2)
    c3 = conv(p2, 4 * n_channels, kernel_size, batchnorm)

    # Up Conv
    kernel = (2,2)
    stride = (2,2)

    u8 = Conv2DTranspose(2 * n_channels, kernel, strides=stride, padding='same')(c3)
    u8 = concatenate([u8, c2])
    c8 = conv(u8, 2 * n_channels, kernel_size, batchnorm)
    u9 = Conv2DTranspose(n_channels, kernel, strides=stride, padding='same')(c8)
    u9 = concatenate([u9, c1], axis=3)
    c9 = conv(u9, n_channels, kernel_size, batchnorm)

    outputs = Conv2D(1, (1, 1), activation='sigmoid')(c9)
    model = Model(inputs=[input], outputs=[outputs])
    return model
```

Detail of training and testing neural net.

```python
def train_model(model, path_train, augment=False,n_epochs = 100,
save_path='./weights/weight.h5', batch_size = 30, learning_rate=0.01,
momentum=0.9, loss="binary_crossentropy"):
    train_dataset = dataset.CatDataset(path_train, im_height, im_width)
    if augment:
        train_dataset.augment()
    X_train, y_train = train_dataset.X, train_dataset.Y

    model.compile(optimizer=Adam(learning_rate=learning_rate), loss=loss,
    metrics=["accuracy"])
    callbacks = [
        EarlyStopping(patience=10, verbose=1),
        ReduceLROnPlateau(factor=0.1, patience=3, min_lr=0.00001, verbose=1),
        ModelCheckpoint(save_path, monitor='loss', mode='min',
                        verbose=1, save_best_only=True, save_weights_only=True)
    ]
    results = model.fit(X_train, y_train, batch_size=batch_size,
                        epochs=n_epochs, callbacks=callbacks,validation_split=0.3)

    return results

def test_model(weight_path, loss, threshold, path_test):
    threshold = threshold
    test_dataset = dataset.CatDataset(path_test, im_width, im_height)
    X_test, y_test = test_dataset.X, test_dataset.Y
    model.load_weights(weight_path)
    pred_test = model.predict(X_test, verbose=1)
    # pred_test= np.interp(pred_test, (pred_test.min(), pred_test.max()), (0, 255.0))

    for i in range(pred_test.shape[0]):
        test_pred = pred_test[i]
        test_pred[test_pred>threshold] = 255
        test_pred[test_pred<=threshold] = 0
        save_img("./run2/pred_{}.jpg".format(i), test_pred)

    model.compile(optimizer=Adam(learning_rate=learning_rate), loss=loss,
    metrics=["accuracy"])
    model.evaluate(X_test, y_test)
```

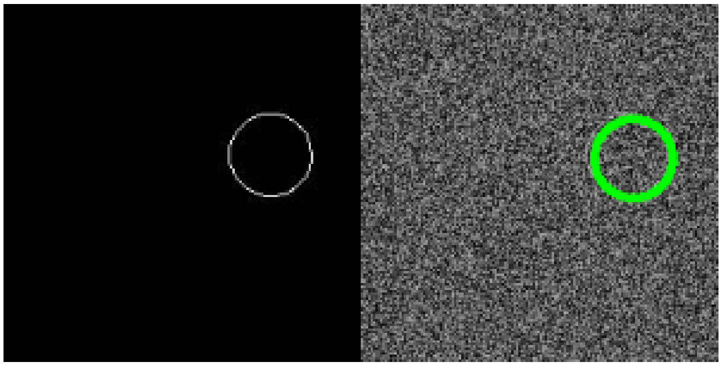
And how the find_circles is implemented.

```python
import cv2 as cv
import os
import numpy as np
from google.colab.patches import cv2_imshow
def find_circles(path, source):
  size = len(os.listdir(path))
  for i in range(size):
    pred_path = path + "pred_{}.jpg".format(i)
    source_path = source + "input.{}.jpg".format(i)
    img = cv.imread(pred_path, 1)
    output =  cv.imread(source_path, 1)

    gray = cv.cvtColor(img, cv.COLOR_BGR2GRAY)
    circles = cv.HoughCircles(gray, cv.HOUGH_GRADIENT,3, 10)
    results = []
    if circles is not None:
      print("circle found")
      circles = np.round(circles[0, :]).astype("int")
      (x,y,r) = circles[0]
      cv.circle(output, (x, y), r, (0, 255, 0), 4)
      results.append((i, circles[0]))
      # cv.imwrite(filepath, output)
    else:
      print("Circle not found")
      results.append(None)
    cv2_imshow(np.hstack([img, output]))

if __name__ == "__main__":
  path = "./data2/"
  source = "./Source/"
  find_circles(path, source)
```

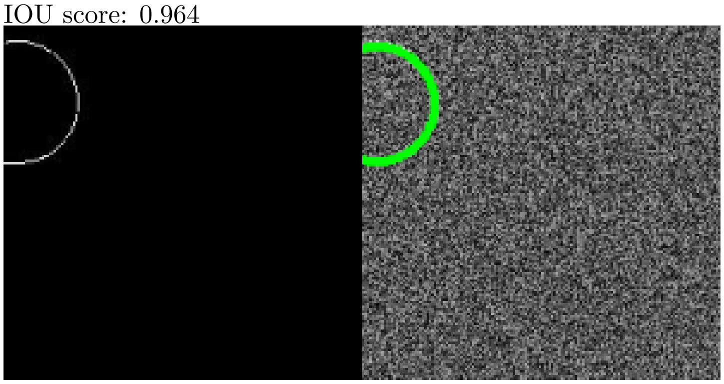
2.3 IOU Optimization

After applying IOU optimization increased the performance of my model, since in a 128 * 128 image only a few pixels are classified as "circle", therefore while using either binary cross entropy and mean square error, the model tends to predict everything as background since predicting as circle will have a MUCH higher chance of making a correct change, and since making wrong prediction penalizes the same for predicting a circle pixel to be background and predicting a background pixel to be circle, and therefore model will tend to predict every pixel to be background and still achieves high 90 accuracy. IOU solves this problem by penalize wrong prediction much higher than usual, and thus the model will tends to find the circle instead of predicting everything to be background.

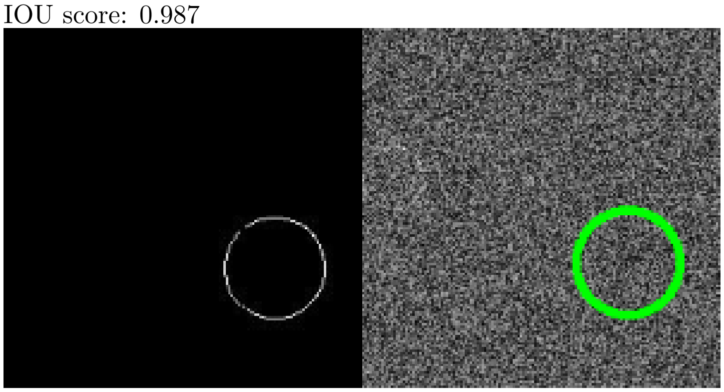
2.4 Visualization and error analysis



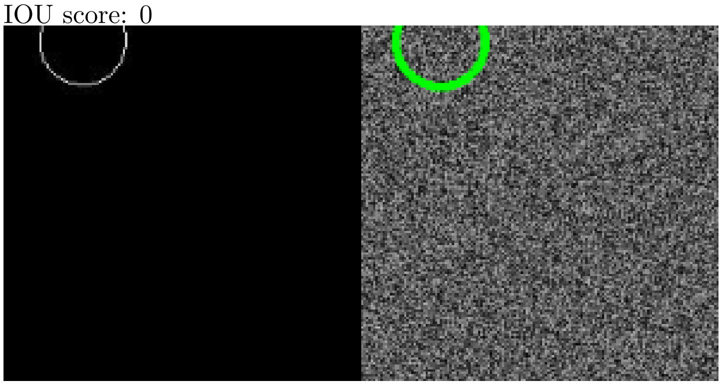
IOU score: 0.964



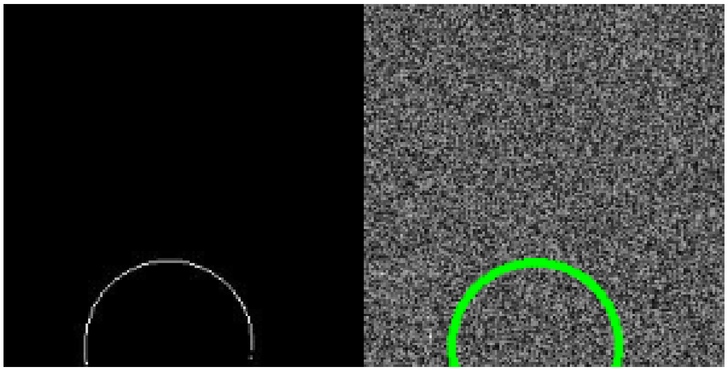
IOU score: 0.987



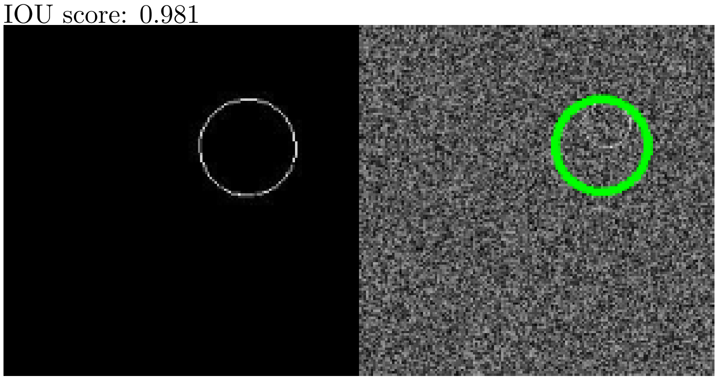
IOU score: 0



IOU score: 0.993



IOU score: 0.981



IOU score: 0.993

As you can see, the third image recieved a IOU score 0 because the neural net made a completely wrong prediction: the circle is on the left side of the image near the edge, but for this one the circle is very hard to find and the neural net may find the noisy pixels have a more circle like pattern and thus made the wrong prediction.

3 Hot Dog or Not Hot Dog

I would say it could be hot dog and it could be not, it could be just some image with similar feature as a hot dog does, or just a t-shirt with a hot dog picture on it. Deep CNN is good at recognizing pattern features but it doesn't necessarily to be a hot dog.