Semantic Segmentation Of Veins and Arteries in Retinal Vasculature

4/20/2023

EGR 452 - Dr. Rhodes

Zachary Lynn

Abstract:

The diagnosis and monitoring of systemic diseases, such as hypertension and diabetes, can be aided by examining retinal vasculature. The microvascular system is particularly relevant in such cases, and the retina is the only place where it can be directly observed. Assessing retinal vessels has been widely used as a proxy for systemic vascular diseases, and recent advances in retinal imaging and computer vision have renewed interest in this field. This project aimed to create a neural network that could segment grayscale images of retinal vasculature into three classes: background, artery, and vein. The algorithm employed an autoencoder architecture with skip connections that spanned across 4 resolutions and 3 kernels. Extensive data augmentation was also used to increase the breadth of the training data. The data was acquired through the Grand Challenge website and was tailored specifically for this purpose. The network was capable of segmenting images of retinal vasculature into three classes (background, artery, and veins) with no pre or post processing. The results achieved in this project did not meet or exceed the results of the currently used methods, however, they still proved that the algorithm could be a viable option if improved in the future. Some results were significantly better than others due to differences in test images. This project did serve its main purpose as a learning experience and introduction to machine learning with an emphasis on image processing.

Introduction:

The retina and its vasculature can be easily observed due to the clear media of the human eye. This provides an opportunity to rapidly and non-invasively image the retina using various techniques, which can offer insights into vital organs like the brain, heart, and kidneys. By analyzing the retinal vasculature, we can monitor and detect changes related to systemic diseases such as hypertension, diabetes, and neurodegenerative disorders. These conditions can lead to severe outcomes like stroke, coronary heart disease, and dementia, which are significant causes of morbidity and mortality in developed countries. Currently, most schemes for classifying retinal vascular changes rely on the subjective human assessment of morphological changes, especially in the early stages of disease.

Researchers believe that changes in the retinal vasculature can provide early biomarkers of disease, even in asymptomatic individuals. For instance, hypertension causes structural changes in vital organ vasculature, including the brain, heart, and kidneys, which are

reflected in the retinal vasculature due to its close functional and pathological link to the cerebral vasculature.

In the past couple of years, many researchers have been working to create diagnostic tools for the retinal vasculature. The STARE dataset for example was created at the University of California, San Diego, to develop an automated diagnosis system for eye diseases. The dataset consists of 20 color retinal images, half of which show pathology that obscures the appearance of retinal blood vessels. The images were taken with a Topcon TRV-50 camera at a 35-degree field of view and are sized at 605 × 700 pixels with 8 bits per RGB color channel[1]. Most other currently used datasets employ color images, however, the RAVIR dataset used infrared images which had greater contrast and overall image quality while causing less discomfort for patients [2].

A common algorithm for biomedical segmentation problems is a neural network architecture known as U-Net. U-Net uses an encoder/decoder architecture with skip connections [3]. Since the unveiling of the U-Net Architecture, mutations of the design have been employed to achieve different segmenting tasks. The architecture proposed in the RAVIR dataset paper is a modification of U-Net that has been shown to outperform the previous versions of U-Net as well as some other popular architectures[2].

Specific Aims

The specific aim of this study is to gain experience working with neural networks for image processing applications. Additionally, this project aimed to achieve a mean dice score of 0.50 or greater. In order to complete the segmentation task the RAVIR dataset paper was studied and their methods were applied to this project. Since I did not have prior experience implementing neural networks, the methods were simplified to make the project more manageable. The differences in architecture will be the exclusion of the auxiliary decoder stream for regularization, fewer convolutional layers, and differing kernel sizes. Data augmentation was also simplified to make the processes easier to understand while learning how to implement these neural network concepts. Different data augmentation techniques and different architectures will be created and tested using the Grand Challenge website in order to compare results and find better algorithms.

The neural network portion of the project was implemented in python3 in order to utilize free libraries such as Keras and OpenCV. Programming in Python3 also provided the ability to train the model on a GPU which was significantly faster than CPU training.

Methods:

The methodology for this project closely resembles the methodology described in the RAVIR Dataset paper[2], but diverges in certain aspects in order to simplify the architecture and downsize the network to be more manageable. The largest difference is that this version does not utilize the auxiliary decoder stream for regularization. The network was implemented in python3 using the Keras interface for TensorFlow. In order to decrease the required training time for networks, the Anaconda environment manager and CUDA tools were used to offload the training workload to a GTX 1070Ti graphics card. This change in the development environment decreased training times by a factor of around ten. This allowed for much more testing to be done and for more variations to be trained and compared.

The network architecture was an autoencoder that covered 4 different resolutions. The network leveraged skip connections from the encoder to the decoder in order to improve the training rate. Dropout layers were used after every convolution, but batch normalization was only used at the input layer, output layer, and in between the encoder and decoder. Relu activation was used for the first layer only, and softmax was used for the output layer; no other layers had activation functions. Figure 1 below shows the network architecture.

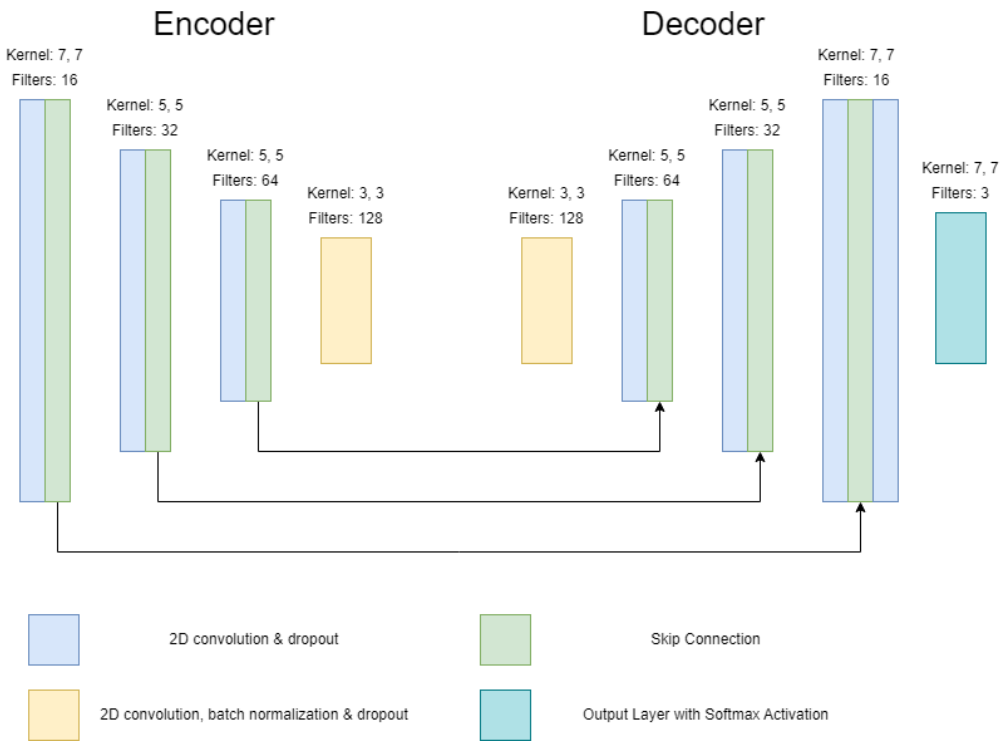

Figure 1: Network Architecture; The network consists of an encoder and decoder that span 3 kernels and four resolutions.

Each convolutional layer on the encoder side utilized a stride of (2,2) in order to reduce the resolution of the input image. Reducing the resolution helped to make the network size smaller and more manageable to train, as well as eliminating some noise in the output by extracting the largest features. On the decoder side, transposed convolution was used with a stride of (2,2) in order to scale the images back up to their original resolution. At the output of the decoder, an additional convolution layer at full resolution was used to learn the final feature classification.

In line with the RAVIR Dataset paper[2], a custom hybrid loss was employed. This loss was a 1:1 weighting of the categorical cross-entropy loss and the dice loss function. The dice loss function examines regional accuracy in an image while the categorical cross-entropy examines pixel-based accuracy, so the hybrid loss was meant to balance these factors. The dice loss that was implemented is given as equation 1 below.

$$L_{Dice} = 1 - \frac{2}{3}\sum_{k=1}^{3}\frac{\sum_{v=1}^{V} G_{v,k}P_{v.k}}{\sum_{v=1}^{V} G_{v,k} + \sum_{v=1}^{V} P_{v,k}},$$

$$where:\ v = pixel,\ k = classes,\ P_{v,k} = predictions,\ G_{v,k} = Truth \quad (1)$$

In order to turn the dice metric into a loss function it was subtracted from 1. Additionally, the dice score was multiplied by two-thirds in order to make it closer in size to the categorical cross-entropy loss. The categorical cross-entropy loss is given below as equation 2.

$$L_{CE} = \frac{-1}{589824}\sum_{v=1}^{589824}\sum_{k=1}^{k} G_{v,k}log(P_{v,k}),$$

$$where:\ v = pixel,\ k = classes,\ P_{v,k} = predictions,\ G_{v,k} = Truth \quad (2)$$

So the final hybrid loss function was simply the sum of the two losses shown below as equation 3.

$$L = 1.0 * L_{Dice} + 1.0 * L_{CE} \quad (3)$$

In order to help generalize and enhance the results of the network, extensive data augmentation was employed. The data augmentation was implemented using MATLAB due to

the integration of image manipulation tools into the MATLAB environment and the familiarity with these tools from previous experience. The data augmentation process first normalized the input images using dynamic contrast stretching based on the histogram of each image. Then the seven nondestructive transforms were applied. Following this, five additional images with random global brightness changes were created by randomly selecting from the pool of previous images. Lastly, two images were blurred using a 5x5 neighborhood averaging filter. Figure A-1 in Appendix A shows an example of the resulting data augmentation from a single input image.

Finally, the algorithm's predictions were made using no pre or post processing of the test images. The predictions were then submitted to the Grand Challenge website for verification. The verification was scored using a few metrics, the most important of which was the dice score. When reading the dice score, 1 represents a perfect match and 0 represents a zero match. Not every result was verified since in some trials the results were visibly worse than in previous attempts.

Results:

There are three sets of important results from the project. The first two results are given to show the progression of the algorithm over time, but most of the emphasis will be on the final results. The first trial that was run used no data augmentation and a constant kernel size of (3,3). The results from this test were underwhelming with a mean dice score of 0.24. The dice score for each test image is given below in Figure 2.

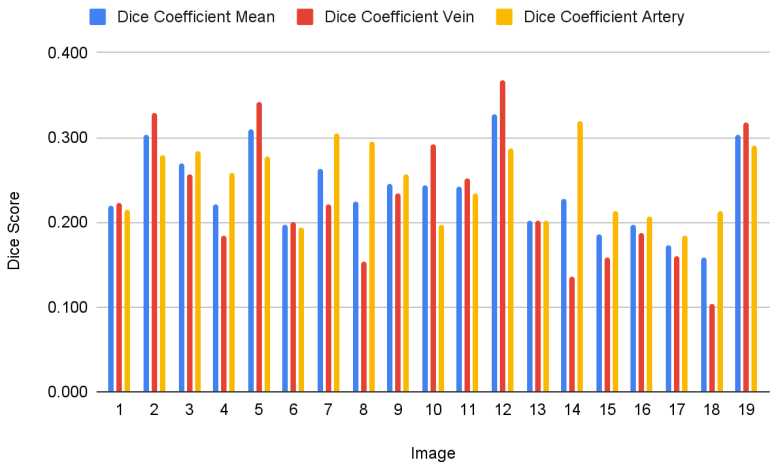

Figure 2: Dice Scores of First Trial; These are the dice scores from the first trial where no data augmentation was used and the kernel was a constant (3,3).

The next important results came after increasing the kernel size to (7,7) in the full-resolution layers and to (5,5) or (3,3) in the lower-resolution layers. Additionally, the data was augmented using the 7 non-destructive transformations. This model was able to achieve a mean dice score of 0.52. The results from this test are given below in Figure 3.

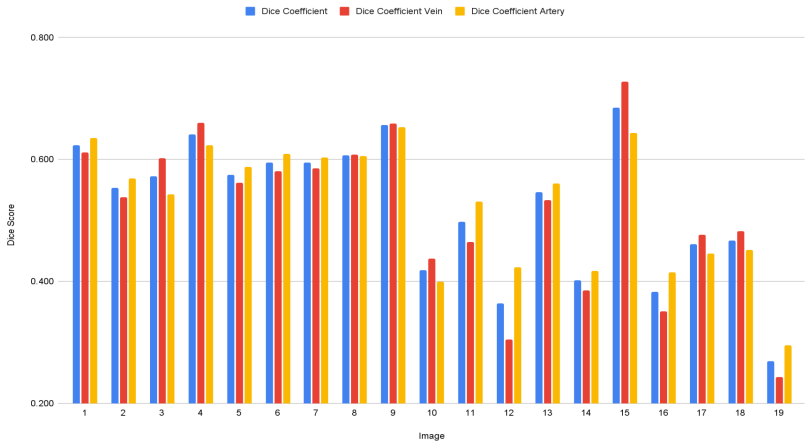

Figure 3: Dice Score of Second Trial; These are the dice scores from the second trial where the data was augmented and the kernel spanned from (7,7) to (3,3).

Now, the final results were achieved by normalizing the training images and adding additional augmentations for brightness variations and blurred images. The final model achieved a mean dice score of 0.63 but had some as high as 0.72 and as low as 0.52. Figure 4 below shows the dice scores for all of the test images.

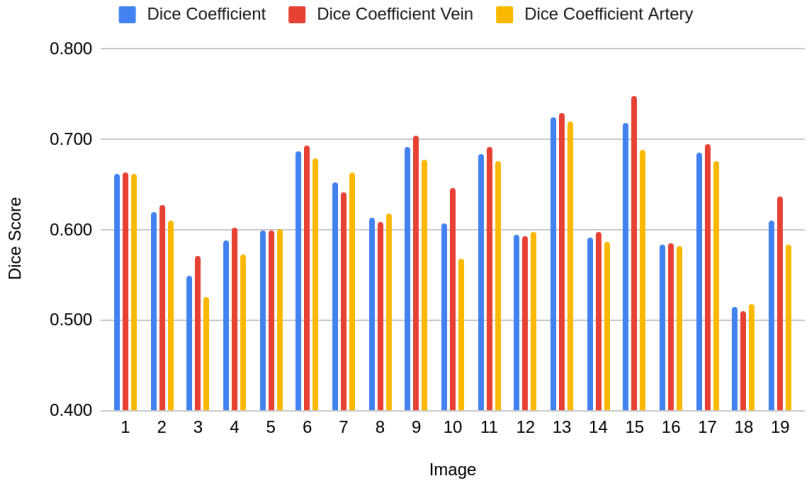

Figure 4: Dice Score of Final Trial; These are the dice scores of the final project output.

The best result from the algorithm was test image 13 with a mean dice score of 0.724. The original test image and the prediction mask are shown below in Figure 5.

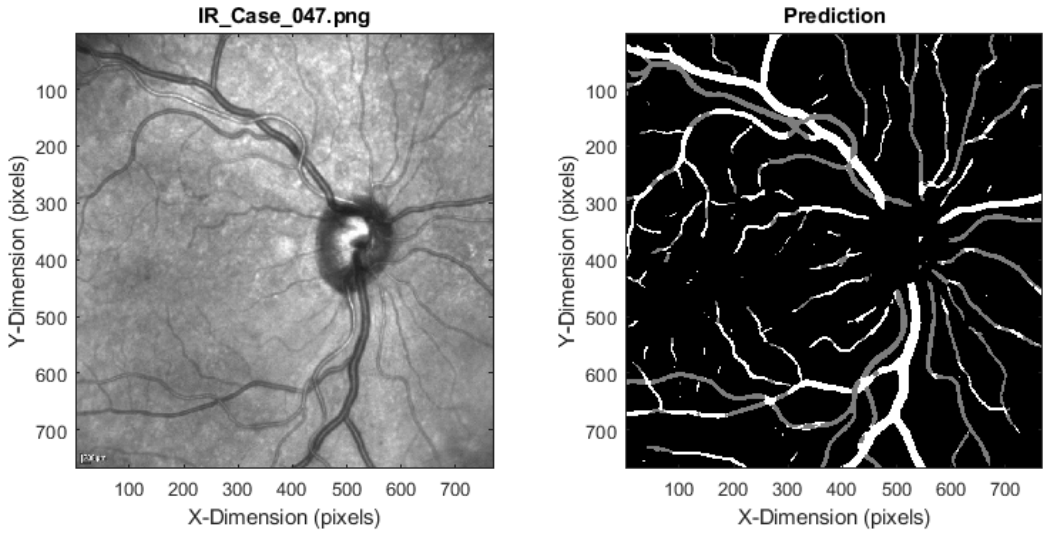

Figure 5: Test Image 13 and The Prediction Maks; This figure shows the original test image on the left and the final prediction mask on the right. White represents veins and gray represents arteries.

The worst result was from test image 18 which scored a mean dice of 0.52. The original test image and the prediction mask for image 18 are shown below in Figure 6.

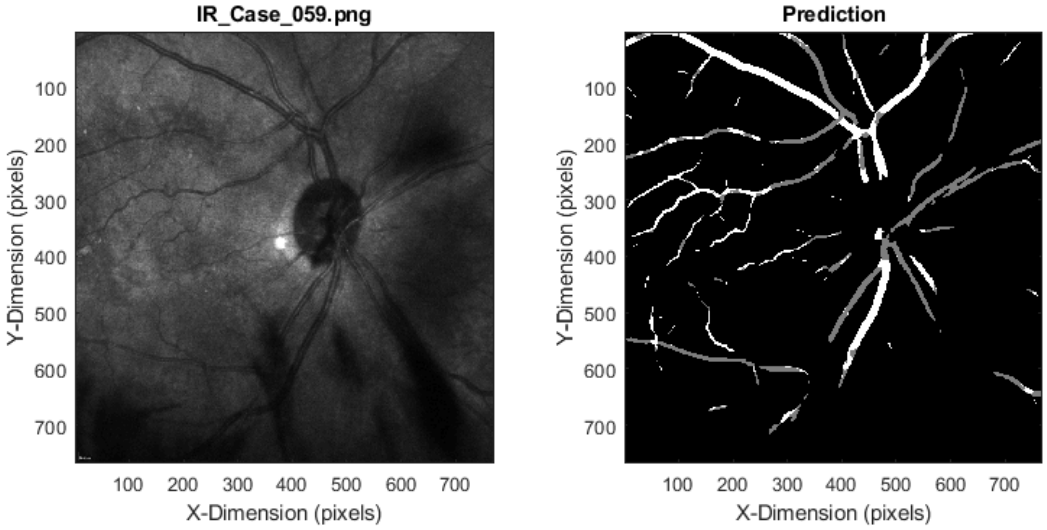

Figure 6: Test Image 18 and The Prediction Maks; This figure shows the original test image on the left and the final prediction mask on the right. White represents veins and gray represents arteries.

Due to the fact that there were 19 test images, the image comparisons of original images and predictions mask will be given in full via an attached zip file. When browsing the images and

noting the dice scores shown in Figure 4, it can be seen that the quality of the original test image had an effect on the results.

Conclusions:

In conclusion, the RAVIR dataset by itself is not large enough to train a general segmentation network, but through the use of data augmentation the data set can be made to generalize fairly well. The RAVIR dataset was designed to include training data with variations in brightness, contrast, and blurring; so after augmentation, the training images are quite diverse. Although the result of this project is not nearly as good as the results achieved in the RAVIR research paper[2], it did confirm that the RAVIR dataset and SEGRAVIR architectures are valid approaches to the problem. Lastly, this project served its main purpose as a learning experience and introduction to neural networks for image processing.

The algorithm did produce two predictions with dice scores above 0.700 and had a mean dice score of 0.63. The images with the lower dice scores tended to be test images with dark splotches, blurring, or low brightness. That being said, researchers have been able to achieve mean dice scores of 0.800 even when considering the less-than-ideal test images. The discrepancy in the results likely came from differences in data augmentation and network architecture. The methods were similar but not identical to those described in the RAVIR dataset research paper[2], so it makes sense they would achieve better results.

In the future, new data augmentation techniques would be evaluated to help improve the network performance. Cropping the original images, then padding them back to the original resolution is a frequently used augmentation technique, but in this project, the results seemed to worsen when using this technique. In the future, I would like to explore this technique in particular. The Python library albumentation also shows a lot of promise, but due to limited time, I was unable to experiment with it. Additionally, in the future, I would try to add in the auxiliary decoder stream for regularization as described in the RAVIR dataset research paper[2].

Acknowledgments:

This project would not have been possible without the RAVIR dataset which was created for the purpose of training neural networks to segment veins and arteries. Additionally, the creators of the dataset published a research paper that gave insights into data augmentation and network architecture.

Additionally, Dr. Denton Bobeldyk from GVSU's Computer Science faculty gave additional direction in the implementation and design of the network architecture.

Bibliography:

[1] A. Hoover, V. Kouznetsova, and M. Goldbaum, "Locating blood vessels in retinal images by piecewise threshold probing of a matched filter response," IEEE Transactions on Medical Imaging, vol. 19, no. 3, pp. 203–210, 2000.

[2] Hatamizadeh, A., Hosseini, H., Patel, N., Choi, J., Pole, C. C., Hoeferlin, C. M., Schwartz, S. D., & Terzopoulos, D. (2022). Ravir: A dataset and methodology for the semantic segmentation and quantitative analysis of retinal arteries and veins in infrared reflectance imaging. *IEEE Journal of Biomedical and Health Informatics*, *26*(7), 3272–3283. https://doi.org/10.1109/jbhi.2022.3163352

This journal article was attached to the challenge posting and provided methodology and a dataset for the purpose of segmenting retinal vasculature.

[3] O. Ronneberger, P. Fischer, and T. Brox, "U-Net: Convolutional networks for biomedical image segmentation," in International Conference on Medical Image Computing and Computer-Assisted Intervention. Springer, 2015, pp. 234–241.

Appendix A:

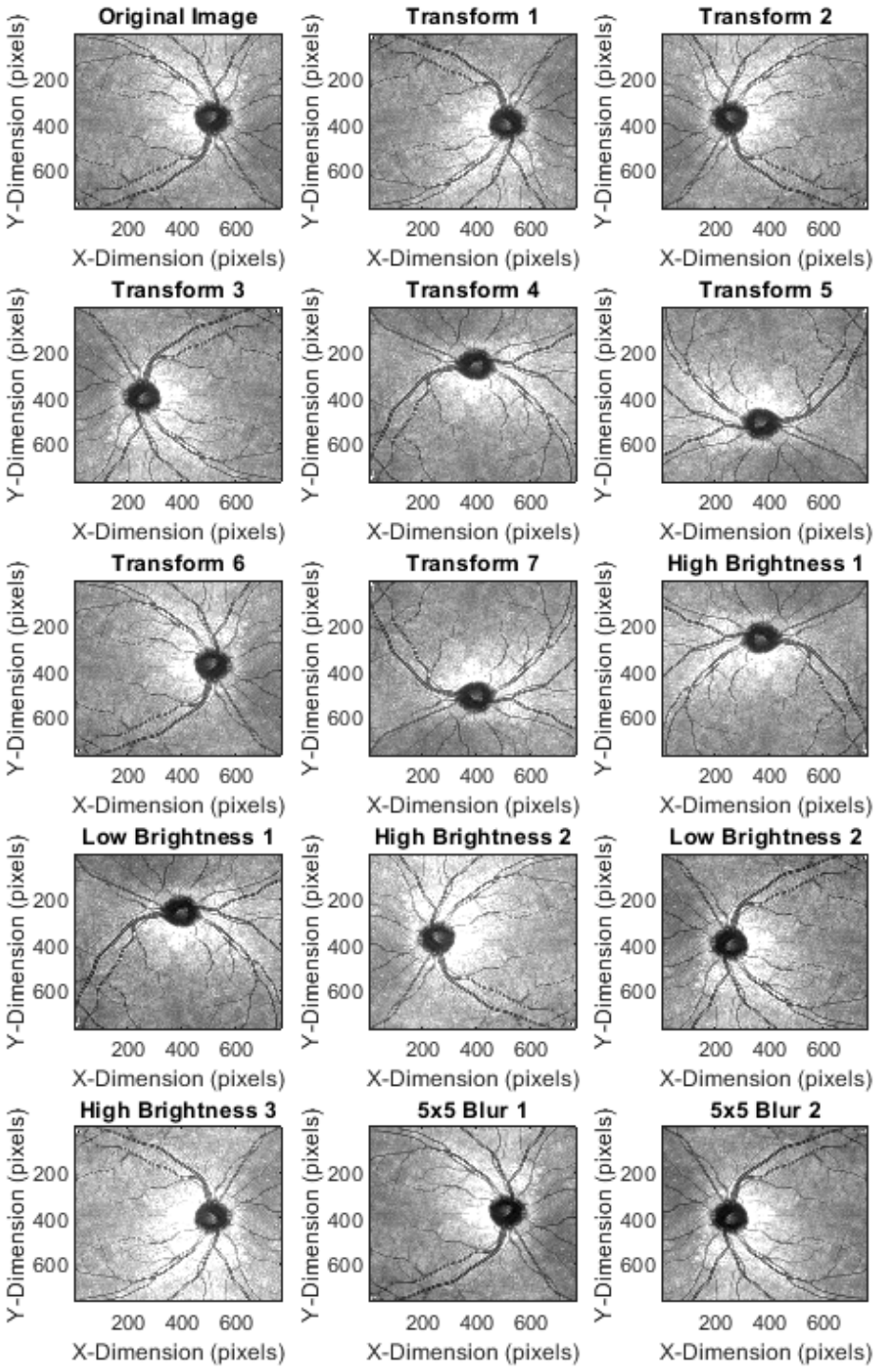

Figure A-1: Data Augmentation Example; Augmented Data Resulting From a Single Input Image.

Appendix B:

B-1: Data Augmentation

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Title: Final Project - Data Augmentation
% Filename: DataAugmentation.m
% Author: Zac Lynn
% Date: 4/2/2023
% Instructor: Dr. Rhodes
% Description: This file performs normalization and data augmentation for
%              the supplied input images. Data is augmented using the
%              7 non-destructive transforms, global brightness
%              changes and 5x5 neighborhood blurring.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
trainingDir = './RAVIRDataset/train/training_images/*';
Files = dir(trainingDir);

for k=3:length(Files)
    fprintf("\nOpening Files: %s\n", Files(k).name);

    img = im2double(imread(strcat("./RAVIRDataset/train/training_images/" + Files(k).name)));
    mask = imread(strcat("./RAVIRDataset/train/training_masks/" + Files(k).name));

    newImg = im2uint8(normalize(img));
    saveImages(newImg, mask, Files(k).name);

    [newImg, newMask] = createNewImages(im2double(newImg), mask);
    normImg = zeros(size(newImg));

    for i=1:size(newImg,3)
        normImg(:,:,i) = im2uint8(normalize(newImg(:,:,i)));
    end
```

```
    saveImages(normImg, newMask, Files(k).name);

end

function [newImg, newMask] = createNewImages(img, mask)
    numFlips = 3;
    numRotates = 5;
    numBrightness = 4;
    numBlur = 2;

    newImg = zeros(size(img, 1), size(img, 2), numRotates + numFlips + numBrightness +
numBlur);
    newMask = zeros(size(mask, 1), size(mask, 2), numRotates + numFlips + numBrightness +
numBlur);

    % Image flips on each axis
    newImg(:,:,1) = flip(img, 1);
    newMask(:,:,1) = flip(mask, 1);

    newImg(:,:,2) = flip(img, 2);
    newMask(:,:,2) = flip(mask, 2);

    newImg(:,:,3) = flip(flip(img, 1), 2);
    newMask(:,:,3) = flip(flip(mask, 1), 2);

    % flip 1 rotates
    newImg(:,:,4) = imrotate(newImg(:,:,1), 90, 'bicubic', 'crop');
    newMask(:,:,4) = imrotate(newMask(:,:,1), 90, 'bicubic', 'crop');

    % flip 2 rotates
    newImg(:,:,5) = imrotate(newImg(:,:,2), 90, 'bicubic', 'crop');
    newMask(:,:,5) = imrotate(newMask(:,:,2), 90, 'bicubic', 'crop');
```

```
% flip 3 rotates
newImg(:,:,6) = imrotate(newImg(:,:,3), 180, 'bicubic', 'crop');
newMask(:,:,6) = imrotate(newMask(:,:,3), 180, 'bicubic', 'crop');
newImg(:,:,7) = imrotate(newImg(:,:,3), 90, 'bicubic', 'crop');
newMask(:,:,7) = imrotate(newMask(:,:,3), 90, 'bicubic', 'crop');
newImg(:,:,8) = imrotate(newImg(:,:,3), 270, 'bicubic', 'crop');
newMask(:,:,8) = imrotate(newMask(:,:,3), 270, 'bicubic', 'crop');

% select 4 random images to make brightness variations of
ind = randperm(numRotates+numFlips);
ind = ind(1:numBrightness);

% Change image brightness
for imageIndex=1:numBrightness
   if (mod(imageIndex, 2) == 0)
      brightness = 1 + (rand() * (15) + 5) / 100.0;
      newImg(:,:,8+imageIndex) = newImg(:,:,ind(imageIndex)) .* brightness + 0.06;
   else
      brightness = 1 - (rand() * (15) + 5) / 100.0;
      newImg(:,:,8+imageIndex) = newImg(:,:,ind(imageIndex)) .* brightness - 0.06;
   end

   newMask(:,:,8+imageIndex) = newMask(:,:,ind(imageIndex));
end

avgMask = ones(3,3);
temp = im2double(newImg(:,:, 11));

img = im2double(img);
for x=1:size(img,2)
   for y=1:size(img,2)
      newImg(y,x,13) = avgNeighbors(img, avgMask, x, y);
      newImg(y,x,14) = avgNeighbors(temp, avgMask, x, y);
   end
```

```
    end

    newMask(:,:,13) = mask;
    newMask(:,:,14) = newMask(:,:,11);
end

function normImg = normalize(img)
    img = im2double(img);
    normImg = zeros(size(img));
    hist = imhist(img);
    cdf = calcCDF(hist);

    lowerBound = 0.00;
    upperBound = 0.9675;

    lower = 999;
    upper = 0;

    % Calculate lower and upper bounds that correspond to percentage bounds
    for i=1:size(cdf)
      if (cdf(i) >= lowerBound && lower == 999)
          lower = i / 255.0;
      elseif (cdf(i) >= upperBound && upper == 0)
          upper = i / 255.0;
      end
    end

    % Apply bounds
    for y=1:size(img,1)
       for x=1:size(img,2)
          if (img(y,x) <= lower)
              normImg(y, x) = 0;
          elseif (img(y,x) >= upper)
              normImg(y, x) = 1.0;
```

```
            else
                normImg(y, x) = (img(y, x) - lower) / upper;
            end
        end
    end
end

function cdf=calcCDF(hist)
    cdf = zeros(size(hist));
    total = sum(hist);

    for i=1:size(hist,1)
        for j = 1:i
            cdf(i) = cdf(i) + hist(j);
        end
        cdf(i) = cdf(i) / total;
    end
end

% Save images
function saveImages(images, masks, name)
    if (ndims(images) == 2)
        if (ndims(masks) == 2)
            filename = name;
            imwrite(images(:,:), gray, strcat("./autoencoderDataBE/train/training_images/",
filename));
            imwrite(masks(:,:), gray, strcat("./autoencoderDataBE/train/training_masks/", filename));
        else
            disp("Image and Mask should be same size");
        end
    else
        for file = 1:size(images, 3)
            filename = strcat(extractBetween(name, "", "."), "_", int2str(file), ".png");
```

```
        imwrite(images(:,:,file), gray, strcat("./autoencoderDataBE/train/training_images/",
filename));
        imwrite(masks(:,:,file), gray, strcat("./autoencoderDataBE/train/training_masks/",
filename));
      end
   end
end
```

B-2: Average Neighborhood

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Title: Final Project - Average Neighbors
% Filename: avgNeighbors.m
% Author: Zac Lynn
% Date: 4/2/2023
% Instructor: Dr. Rhodes
% Description: This file contains code that can apply neighborhood averages
%               given an image, a mask to apply the averaging, and
%               values for i and j on which to center the mask. For a
%               basic average the mask should be all ones. For
%               laplacians, use the mask accordingly.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [avg, total]=avgNeighbors(img, mask, i, j)
% Uses the supplied mask to average the neighbors of the pixel found at
% (i, j)

   sum = 0.0;
   count = 0;

   yUnder = 0; % could use to apply values to the pixels that are out of bounds
   yOver = 0;
   xUnder = 0;
```

```
xOver = 0;

yMin = j - floor(size(mask, 1) / 2);
if (yMin < 1)
  yUnder = abs(yMin) + 1;
  yMin = 1;
end

yMax = j + floor(size(mask, 1) / 2);
if  (yMax > size(img, 1))
   yOver = yMax - size(img, 1);
   yMax = size(img, 1);
end

xMin = i - floor(size(mask, 2) / 2);
if (xMin < 1)
  xUnder = xMin - 1;
  xMin = 1;
end

xMax = i + floor(size(mask, 2) / 2);
if  (xMax > size(img, 2))
   xOver = xMax - size(img, 2);
   xMax = size(img, 2);
end

for x = xMin:xMax
   for y = yMin:yMax
      sum = sum + img(y, x) * mask(yMax - y + 1, xMax - x + 1);
      count = count + mask(yMax - y + 1, xMax - x + 1);
   end
end

avg = sum / count;
```

```
    total = sum;
end
```

B-3: Network Training

```
################################################################################
# Title: Final Project - Encoder Network
# Filename: encoder.m
# Author: Zac Lynn
# Date: 4/2/2023
# Instructor: Dr. Rhodes
# Description: This code defines the encoder architecture and trains the
#              network. The Learning rate and Batch size should be
#              updated at periods of 50 epochs.
################################################################################
import tensorflow as tf
from tensorflow import keras
import tensorflow.keras.backend as K
import numpy as np
import matplotlib.pyplot as plt
import cv2
import os

global training_image_batch, training_mask_batch
global autoencoder

img_x = 768
img_y = 768

# Learning rate = 0.001 * 0.5 ^ epoch / 50
LR = 0.001 * 0.5 ** 0
print("LEARNING RATE: " + str(LR), end="\n\n\n")

def hybridLoss(y_true, y_pred):
    y_true_f = K.flatten(y_true)
```

```
    # Create a tensor for each prediction plane. Values are the raw predictions
    y_pred_class1, y_pred_class2, y_pred_class3 = tf.split(y_pred, num_or_size_splits=3,
axis=-1)
    y_pred_class1 = K.flatten(tf.squeeze(y_pred_class1, axis=-1))
    y_pred_class2 = K.flatten(tf.squeeze(y_pred_class2, axis=-1))
    y_pred_class3 = K.flatten(tf.squeeze(y_pred_class3, axis=-1))

    # Create a tensor for each class. Tensors values are 1 or 0
    y_true_class1 = tf.where(tf.equal(y_true_f, 0), tf.ones_like(y_true_f), tf.zeros_like(y_true_f))
    y_true_class2 = tf.where(tf.equal(y_true_f, 1), tf.ones_like(y_true_f), tf.zeros_like(y_true_f))
    y_true_class3 = tf.where(tf.equal(y_true_f, 2), tf.ones_like(y_true_f), tf.zeros_like(y_true_f))

    intersection1 = K.sum(y_true_class1 * y_pred_class1)
    intersection2 = K.sum(y_true_class2 * y_pred_class2)
    intersection3 = K.sum(y_true_class3 * y_pred_class3)

    pixelSum1 = K.sum(y_pred_class1)
    pixelSum2 = K.sum(y_pred_class2)
    pixelSum3 = K.sum(y_pred_class3)

    dice1 = (intersection1) / (pixelSum1 + K.sum(y_true_class1))
    dice2 = (intersection2) / (pixelSum2 + K.sum(y_true_class2))
    dice3 = (intersection3) / (pixelSum3 + K.sum(y_true_class3))

    diceLoss = 1.0 - (2.0 / 3.0) * (dice1 + dice2 + dice3)



    ceLoss = K.sparse_categorical_crossentropy(y_true, y_pred, from_logits=False)
    ceLoss = K.mean(ceLoss)

    return 1.0 * diceLoss  + 1.0 * ceLoss

def read_training_data():
```

```
    global training_image_batch, training_mask_batch
    classes = {"background": 0, "artery": 128, "vein": 255}

    training_image_dir = "./autoencoderDataBE/train/training_images/"
    training_mask_dir = "./autoencoderDataBE/train/training_masks/"

    training_image_filenames = os.listdir(training_image_dir)

    # Load the training images
    training_image_batch = []
    for file in training_image_filenames:
        training_image_batch.append(cv2.imread(training_image_dir + file, 0) / 255.0)
    training_image_batch = np.array(training_image_batch, dtype=np.float32)

    # Load the training masks
    training_mask_batch = []
    for file in training_image_filenames:
        training_mask_batch.append(cv2.imread(training_mask_dir + file, 0))
    training_mask_batch = np.array(training_mask_batch, dtype=np.float32)

    # Threshold any mask values that changed with image rotation
    training_mask_batch[(training_mask_batch == classes["artery"])] = 1
    training_mask_batch[(training_mask_batch == classes["vein"])] = 2

    training_mask_batch[(training_mask_batch == classes["artery"]-1)] = 1
    training_mask_batch[(training_mask_batch == classes["vein"]-1)] = 2

def define_model():
    global autoencoder

    encoder_input = keras.Input(shape=(img_x, img_y, 1), name="image_in")

    # ---------- INPUT ----------
    x = keras.layers.Conv2D(16, (7, 7), activation='relu', padding='same') (encoder_input)
```

```
block1 = keras.layers.BatchNormalization()(x)

# ---------- ENCODER----------
x = keras.layers.Conv2D(32, (7, 7), strides=(2,2), padding='same') (block1)
block2 = keras.layers.Dropout(0.15)(x)

x = keras.layers.Conv2D(64, (5, 5), strides=(2,2), padding='same') (block2)
block3 = keras.layers.Dropout(0.15)(x)

x = keras.layers.Conv2D(128, (3, 3), strides=(2,2), padding='same') (block3)
block4 = keras.layers.Dropout(0.15)(x)
block4 = keras.layers.BatchNormalization()(block4)  #??

encoder = keras.Model(encoder_input, block4)

# ---------- DECODER----------
decoder_input = keras.layers.Conv2D(128, (3, 3), padding='same')(block4)
x = keras.layers.Dropout(0.15)(decoder_input)
x = keras.layers.BatchNormalization()(x)          #??

x = keras.layers.Conv2D(128, (3, 3), padding='same')(x)
x = keras.layers.add([x, block4])
x = keras.layers.Dropout(0.15)(x)

x = keras.layers.Conv2DTranspose(64, (5, 5), strides=(2, 2), padding='same') (x)
x = keras.layers.add([x, block3])
x = keras.layers.Dropout(0.15)(x)

x = keras.layers.Conv2DTranspose(32, (7, 7), strides=(2, 2), padding='same') (x)
x = keras.layers.add([x, block2])
x = keras.layers.Dropout(0.15)(x)

x = keras.layers.Conv2DTranspose(16, (7, 7), strides=(2, 2), padding='same') (x)
x = keras.layers.add([x, block1])
```

```
    x = keras.layers.Dropout(0.15)(x)

    # ---------- OUTPUT ----------
    x = keras.layers.Conv2D(16, (7, 7), activation='relu', padding='same') (x)
    x = keras.layers.BatchNormalization()(x)
    x = keras.layers.Dropout(0.15)(x)

    x = keras.layers.Conv2D(16, (7, 7), activation='relu', padding='same') (x)
    x = keras.layers.BatchNormalization()(x)

    # Kernel is 1,1 for non blur version
    decoder_output = keras.layers.Conv2D(3, (7, 7), activation='softmax', padding='same') (x)

    opt = keras.optimizers.Adam(lr=LR)

    autoencoder = keras.Model(encoder_input, decoder_output, name="autoencoder")

    autoencoder.compile(opt, loss=hybridLoss)

    autoencoder.summary()


define_model()
read_training_data()

# Load model - only load after first 50 epochs
# autoencoder.load_weights('./model/R4_BE_50')

# Train model
autoencoder.fit(training_image_batch, training_mask_batch, epochs=50, batch_size=3,
shuffle=True)

# Save Model
autoencoder.save_weights("./model/R4_BE_50")
```

```
# Test image
testImg = np.array(cv2.imread("./RAVIRDataset/test/IR_Case_060.png", 0) / 255.0,
dtype=np.float32)
testImg = testImg.reshape(-1, 768, 768, 1)
ex = autoencoder.predict(testImg)[0]


r = np.zeros((img_x, img_y))
g = np.zeros((img_x, img_y))
b = np.zeros((img_x, img_y))


for row in range(len(ex)):
    for col in range(len(ex[row])):
        r[row][col] = ex[row][col][0]
        g[row][col] = ex[row][col][1]
        b[row][col] = ex[row][col][2]

        ex[row][col][0] = 0


plt.figure(1)
plt.imshow(testImg[0], cmap='gray')
plt.title("Original")

plt.figure(2)
plt.imshow(r, cmap='gray')
plt.title("Background")

plt.figure(3)
plt.imshow(g, cmap='gray')
plt.title("Artery")
```

```
plt.figure(4)
plt.imshow(b, cmap='gray')
plt.title("Vein")

plt.figure(5)
plt.imshow(ex, cmap='gray')
plt.title("Prediction")

plt.show()
```

B-4: Network Predictions

```
##############################################################################
# Title: Final Project - Make Predictions
# Filename: MakePredictions.m
# Author: Zac Lynn
# Date: 4/2/2023
# Instructor: Dr. Rhodes
# Description: This code reads in previously trained network weights and
#                 makes predictions based on test images.
##############################################################################
import tensorflow as tf
import tensorflow.keras.backend as K
from tensorflow import keras
import numpy as np
import cv2
import os

global model, test_image_batch, test_image_filenames
img_x = 768
img_y = 768

def hybridLoss(y_true, y_pred):
    y_true_f = K.flatten(y_true)
```

```
    # Create a tensor for each prediction plane. Values are the raw predictions
    y_pred_class1, y_pred_class2, y_pred_class3 = tf.split(y_pred, num_or_size_splits=3,
axis=-1)
    y_pred_class1 = K.flatten(tf.squeeze(y_pred_class1, axis=-1))
    y_pred_class2 = K.flatten(tf.squeeze(y_pred_class2, axis=-1))
    y_pred_class3 = K.flatten(tf.squeeze(y_pred_class3, axis=-1))

    # Create a tensor for each class. Tensors values are 1 or 0
    y_true_class1 = tf.where(tf.equal(y_true_f, 0), tf.ones_like(y_true_f), tf.zeros_like(y_true_f))
    y_true_class2 = tf.where(tf.equal(y_true_f, 1), tf.ones_like(y_true_f), tf.zeros_like(y_true_f))
    y_true_class3 = tf.where(tf.equal(y_true_f, 2), tf.ones_like(y_true_f), tf.zeros_like(y_true_f))

    intersection1 = K.sum(y_true_class1 * y_pred_class1)
    intersection2 = K.sum(y_true_class2 * y_pred_class2)
    intersection3 = K.sum(y_true_class3 * y_pred_class3)

    pixelSum1 = K.sum(y_pred_class1)
    pixelSum2 = K.sum(y_pred_class2)
    pixelSum3 = K.sum(y_pred_class3)

    dice1 = (intersection1) / (pixelSum1 + K.sum(y_true_class1))
    dice2 = (intersection2) / (pixelSum2 + K.sum(y_true_class2))
    dice3 = (intersection3) / (pixelSum3 + K.sum(y_true_class3))

    diceLoss = 1.0 - (2.0 / 3.0) * (dice1 + dice2 + dice3)



    ceLoss = K.sparse_categorical_crossentropy(y_true, y_pred, from_logits=False)
    ceLoss = K.mean(ceLoss)

    return 1.0 * diceLoss  + 1.0 * ceLoss

def define_model():
```

```
global autoencoder

encoder_input = keras.Input(shape=(img_x, img_y, 1), name="image_in")

# ---------- INPUT ----------
x = keras.layers.Conv2D(16, (7, 7), activation='relu', padding='same') (encoder_input)
block1 = keras.layers.BatchNormalization()(x)

# ---------- ENCODER----------
x = keras.layers.Conv2D(32, (7, 7), strides=(2,2), padding='same') (block1)
block2 = keras.layers.Dropout(0.15)(x)

x = keras.layers.Conv2D(64, (5, 5), strides=(2,2), padding='same') (block2)
block3 = keras.layers.Dropout(0.15)(x)

x = keras.layers.Conv2D(128, (3, 3), strides=(2,2), padding='same') (block3)
block4 = keras.layers.Dropout(0.15)(x)
block4 = keras.layers.BatchNormalization()(block4)  #??

encoder = keras.Model(encoder_input, block4)

# ---------- DECODER----------
decoder_input = keras.layers.Conv2D(128, (3, 3), padding='same')(block4)
x = keras.layers.Dropout(0.15)(decoder_input)
x = keras.layers.BatchNormalization()(x)          #??

x = keras.layers.Conv2D(128, (3, 3), padding='same')(x)
x = keras.layers.add([x, block4])
x = keras.layers.Dropout(0.15)(x)

x = keras.layers.Conv2DTranspose(64, (5, 5), strides=(2, 2), padding='same') (x)
x = keras.layers.add([x, block3])
x = keras.layers.Dropout(0.15)(x)
```

```
    x = keras.layers.Conv2DTranspose(32, (7, 7), strides=(2, 2), padding='same') (x)
    x = keras.layers.add([x, block2])
    x = keras.layers.Dropout(0.15)(x)

    x = keras.layers.Conv2DTranspose(16, (7, 7), strides=(2, 2), padding='same') (x)
    x = keras.layers.add([x, block1])
    x = keras.layers.Dropout(0.15)(x)

    # ---------- OUTPUT ----------
    x = keras.layers.Conv2D(16, (7, 7), activation='relu', padding='same') (x)
    x = keras.layers.BatchNormalization()(x)
    x = keras.layers.Dropout(0.15)(x)

    x = keras.layers.Conv2D(16, (7, 7), activation='relu', padding='same') (x)
    x = keras.layers.BatchNormalization()(x)

    # KErnel is 1,1 for non blur version
    decoder_output = keras.layers.Conv2D(3, (7, 7), activation='softmax', padding='same') (x)

    opt = keras.optimizers.Adam(lr=0.001)

    autoencoder = keras.Model(encoder_input, decoder_output, name="autoencoder")

    autoencoder.compile(opt, loss=hybridLoss)

    autoencoder.summary()

def read_test_data():
    global test_image_batch, test_image_filenames

    test_image_dir =  "./RAVIRDataset/test/"
    test_image_filenames = os.listdir(test_image_dir)

    # Load the training images
```

```
    test_image_batch = []
    for file in test_image_filenames:
        test_image_batch.append(cv2.imread(test_image_dir + file, 0) / 255.0)
    test_image_batch = np.array(test_image_batch, dtype=np.float32)

def make_predictions(output_dir):
    prediction = []
    for image in test_image_batch:
        # Make a prediction with the model
        prediction.append(autoencoder.predict(np.expand_dims(image, axis=0))[0])

    # Threshold predictions
    threshold = 0.01

    for image in range(len(prediction)):
        imageOut = np.zeros((img_x,img_y))
        for row in range(len(prediction[image])):
            for col in range(len(prediction[image][row])):

                if (prediction[image][row][col][0] >= prediction[image][row][col][1] and
prediction[image][row][col][0] >= prediction[image][row][col][2]):
                    # Background is 0
                    imageOut[row][col] = 0
                elif (prediction[image][row][col][1] >= prediction[image][row][col][0] and
prediction[image][row][col][1] >= prediction[image][row][col][2]):
                    if (prediction[image][row][col][1] >= threshold):
                        # Arteries are 128
                        imageOut[row][col] = 128
                    else:
                        imageOut[row][col] = 0

                elif (prediction[image][row][col][2] >= prediction[image][row][col][0] and
prediction[image][row][col][2] >= prediction[image][row][col][1]):
                    if (prediction[image][row][col][2] >= threshold):
```

```
                # Veins are 255
                imageOut[row][col] = 256
            else:
                imageOut[row][col] = 0

    cv2.imwrite(output_dir + test_image_filenames[image], imageOut)

define_model()
autoencoder.load_weights("./model/R4_BE_150")

read_test_data()
make_predictions("./predictions/")
```