Spring 2020 Data Science Institute

Capstone Project Final Report

Cloud-based cup to disc ratio estimator for retinal fundus images

Rohan Bareja, Wadood Chaudhary, Christine Lee, Janet Prumachuk

May 15, 2020

1. Introduction

1.1. Project Scope and Goals

This Capstone project is a collaboration between Columbia University's Data Science Institute and Johnson & Johnson.

The optic cup to optic disc ratio (CDR) is an important measurement in glaucoma diagnosis. In this project the objective was to obtain the CDR, not to make a glaucoma diagnosis.

Cup-to-disc ratio is defined as the ratio of vertical distances between pixels at the highest and lowest vertical position inside the cup and disc region. Evaluation of the CDR is done upon examination of the retina or a retinal (fundus) photo.

CDR measurements are highly subjective and inconsistent even among experts. Current methods are mostly qualitative, resulting in poor reproducibility. While AI models have been developed, none have been deployed on the Google Cloud Platform (GCP) for practical use in a clinical setting.

The goal of this project was to leverage GCP and its AI tools to build a deployable artificial intelligence based system to estimate cup to disc ratios for unannotated color fundus images.

1.2. Data Sources

1.2.1. Retinal Fundus Images for Glaucoma Analysis (RIGA dataset)

The RIGA dataset is an open access source of de-identified Real World Data consisting of retinal fundus images. There are 750 original images and 4500 manually marked images. Some images are in JPEG format and some are in TIFF format.

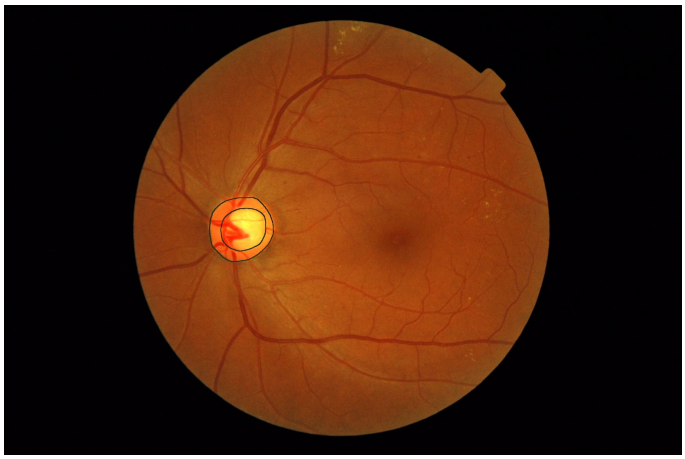


Figure 1. Example of an annotated image. The smaller marked area is the optic cup. The larger marked area is the optic disc. The tab on the upper right side is for orientation and alignment.

The RIGA dataset includes 3 collections:

1. The MESSIDOR collection contains 460 original images and 2760 annotated images.
2. The Bin Rushed Ophthalmic center collection contains 195 original images and 1200 annotated images.
3. The Magrabi Eye center collection contains 95 original images and 570 annotated images.

1.2.2. RIM-ONE and DRISHTI fundus datasets

We explore the use of training datasets beyond the above first proposed dataset RIGA to include other publicly available fundus image datasets, RIM-ONE v3 and DRISHTI.

First, the inclusion of a more diverse combination of training data sources is expected to help our machine learning model capture more variation in the image quality/output across image-taking devices and hopefully enable higher predictive performance when applied on unseen test images.

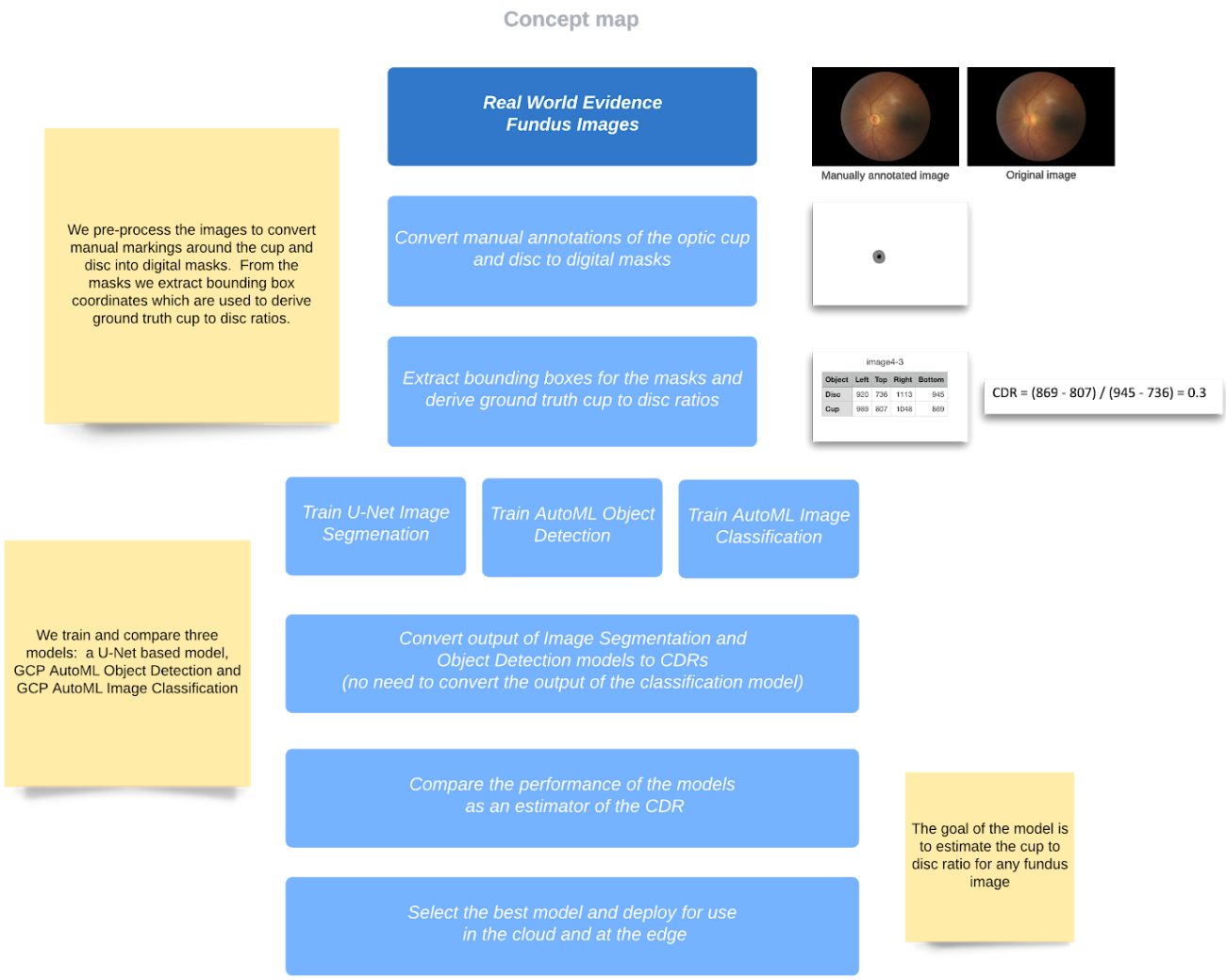
Second, our objective was to identify/use only publicly available fundus image datasets that contain ground truth segmentation for our segmentation prediction model, in other words the ground truth "masks" of the cup and of the disc. We realized that there are a number of fundus image datasets that do not contain such ground truth segmentation, which would be unfeasible dataset candidates for our segmentation modeling project. We found two datasets that meet the above criteria.

Dataset **"RIM ONE-v3"**, from the MIAG group of the University of La Laguna in Spain, consists of 159 fundus images which have been labeled by expert ophthalmologists for both disc and cup. Dataset **"DRISHTI-GS"**, from the Aravind Eye hospital in Madurai, India consists of 101 fundus images also labeled for disc and cup.

1.2.3. REFUGE Dataset

This is the most complete, cleanest and robust dataset available. The dataset is divided into two sets: training (400 images) and validation (400 images). The 400 images of the training set are in JPG format. The training set also includes 400 corresponding ground truth segmentations in BMP format. All images have a size of 2124 × 2056.

1.3. Concept Slide



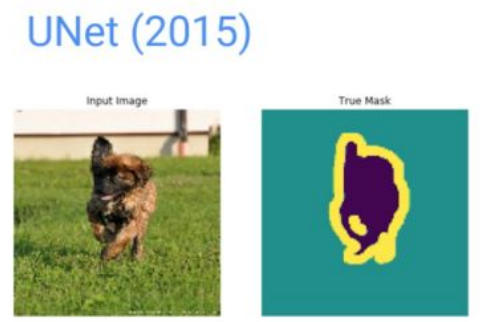
2. Modelling Approaches

2.1. U-Net CNN model for Image Segmentation

The goal of image segmentation is to label each pixel of an image with a corresponding class of what is being represented. The result is pixel-level image classification where the output itself is a high resolution image, for which each pixel in the image is classified to a particular class, such as belonging to the mask or not belonging to the mask of the object of interest. This task is sometimes referred to as semantic image segmentation.

U-Net is a modified convolutional neural network that was developed for biomedical image segmentation at the Computer Science Department of the University of Freiburg, Germany. While a convolutional neural network focuses on image classification tasks, U-Net, based on CNNs but with modifications and extensions to its architecture, is meant to find and detect localizations of disease and to yield more precise segmentations. It can also be termed as an object detection network, developed primarily for biomedical images.

We explored image segmentation with a lighter U-Net model, which has a simpler architecture as compared to an earlier adoption of a deeper U-Net model.



https://www.tensorflow.org/beta/tutorials/images/segmentation
U-Net: Convolutional Networks for Biomedical Image Segmentation

2.1.1. U-Net Model Architecture Overview

As obvious from the name, the model architecture has a U-shape. Similar to the original U-Net, the U-Net for fundus segmentation is symmetric and consists of a contracting path (left side) and an expansive path (right side).

The contracting path is composed of multiple blocks, each block having convolution layers, often followed by pooling layers. In the expansive path, the transformed feature maps will be upsized to an output in the image's original size.

2.1.1.1 Convolution layers and pooling layers

A convolutional neural network (CNN) consists of an input and an output layer, as well as multiple hidden layers. The hidden layers of a CNN typically consist of a series of convolutional layers.

Convolution layers are key components in performing most of the feature extraction in a CNN by using a set of learnable filters. These filters are used to convolve across the width and height of the input image and, at each unique position, computes the element-wise multiplication between the input matrix and the filter/kernel matrix. The result of this operation is a 2-dimensional feature map for each filter applied. In other words, convolutional layers convolve across the input (image) and pass its result to the next layer, which might be another convolution layer or a pooling layer.

The purpose of using pooling layers in a CNN is to reduce the spatial size of the feature maps. *Figure 3* illustrates how a pooling layer reduces or down-samples the input from 224x224 pixels to 112x112 pixels. Pooling reduces the number of parameters and the computation in the neural network, and hence, helps mitigate model overfitting.

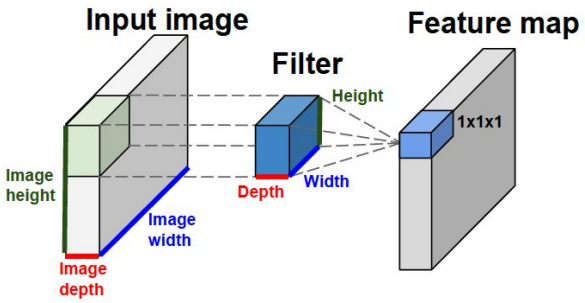


Figure 2. Example of a convolution layer. A feature map is generated by one filter when applied to an input image.

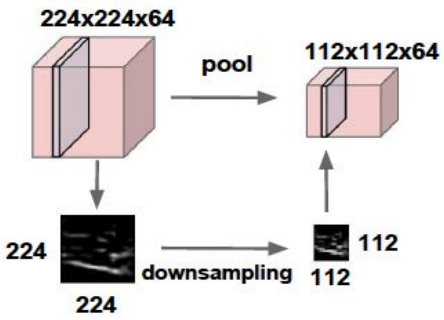


Figure 3. Example of a pooling layer.

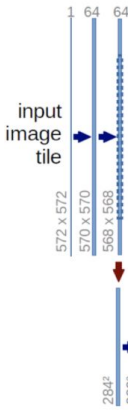


Figure 4. Above is a single convolution block from the U-Net contracting path, which consists of two convolutional layers with relu activation for non-linearity, followed by a max-pooling layer.

The architecture of an image segmentation U-Net model is illustrated in *Figure 5* below.

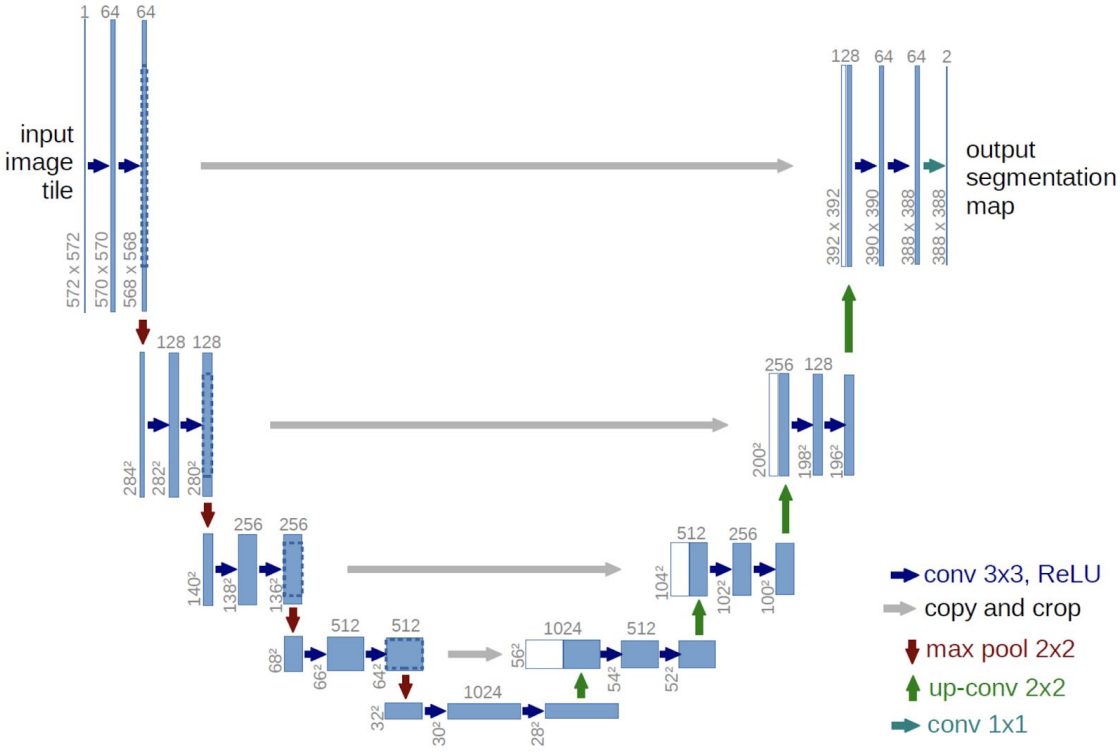


Figure 5. Architecture of a U-Net neural network

Munir, K., Elahi, H., Ayub, A., Frezza, F., & Rizzi, A. provides an explanation of the two paths for a U-Net model [3]:

"The first contraction path is known as an encoder, which captures the context in the image. This is mainly a stack of convolution and pooling layers. The second path is the symmetric expanding path, also known as a decoder, which uses transposed convolutions and enables the precise localization. It is an end-to-end FCN network with no dense layer, only convolutional layers. Therefore, it can accept an image of any size."

Ronneberger, Fischer, and Brox's original description of the U-Net architecture is as follows [2]:

"It consists of a contracting path (left side) and an expansive path (right side). The contracting path follows the typical architecture of a convolutional network. It consists of the repeated application of two 3x3 convolutions (unpadded convolutions), each followed by a rectified linear unit (ReLU) and a 2x2 max pooling operation with stride 2 for downsampling. At each downsampling step we double the number of feature channels. Every step in the expansive path consists of an upsampling of the feature map followed by a 2x2 convolution ("up-convolution") that halves the number of feature channels, a concatenation with the correspondingly cropped feature map from the contracting path, and two 3x3 convolutions, each followed by a ReLU. The cropping is necessary due to the loss of border pixels in every convolution. At the final layer a 1x1 convolution is used to map each 64-component feature vector to the desired number of classes. In total the network has 23 convolutional layers."

2.1.2. Why use U-Net for image segmentation

The up-sampling (right-hand) path of a U-Net captures spatial or localization information ("WHERE"), and combines it with feature information ("WHAT") from the (left-hand) contracting path. The output from the expanding path at each level is concatenated with the contracting path to get information from feature maps learnt, to yield more precise information necessary for segmentation.

In a typical convolution network, at each step there is a reduction in height and width of image because of pooling, to learn contextual information ("WHAT"). We may lose the "WHERE" in a typical CNN, and hence the U-Net model is beneficial for biomedical image segmentation tasks where both "WHAT" and "WHERE" are needed and are captured.

Due to the above reasons, U-Nets often require fewer training data, and provide better segmentation results compared to other segmentation modeling techniques.

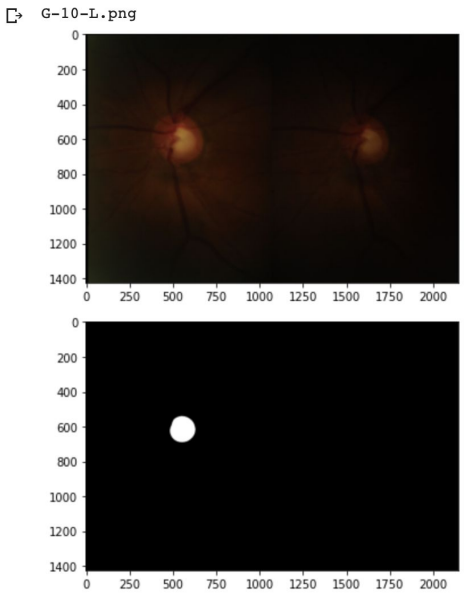
2.1.3. Image Data Pre-Processing

Our pre-processing for U-Net includes two steps:

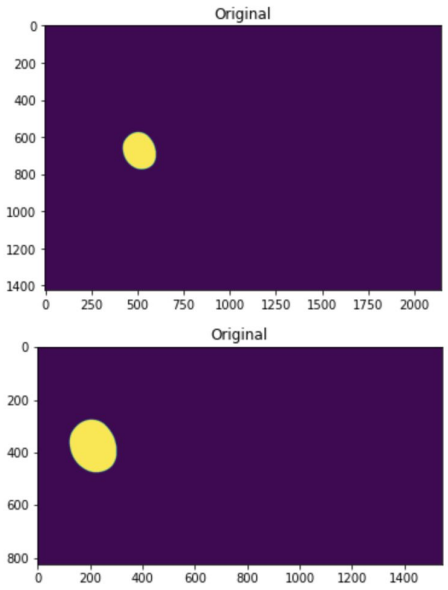
Cropping : We cropped the image by 300 px from top and bottom, as well as left and right.

```
[ ] # Setting the points for cropped image
    from PIL import Image
    from keras.preprocessing.image import save_img,img_to_array
    def crop(data):

      res_img = []
      for i in range(len(data)):
        #print(len(data))
      # Cropped image of above dimension
      # (It will not change orginal image)
        filename = os.path.basename(data[i])
        filename = os.path.splitext(filename)[0]
        im = imread(data[i])
        im1 = im[300:(im.shape[0]-300),300:(im.shape[1]-300)]
        img_array = img_to_array(im1)
        save_img('/content/gdrive/My Drive/Drishti_RIM_Cropped_Data/' + filename + '.png', img_array)
        #print(im1)
        res_img.append(im1)

      # Shows the image in image viewer
      original = res_img[1]
      display_one(imread(data[1]))
      display_one(original)
```

Original Image and its cup mask is seen below:



After cropping, here is the mask.

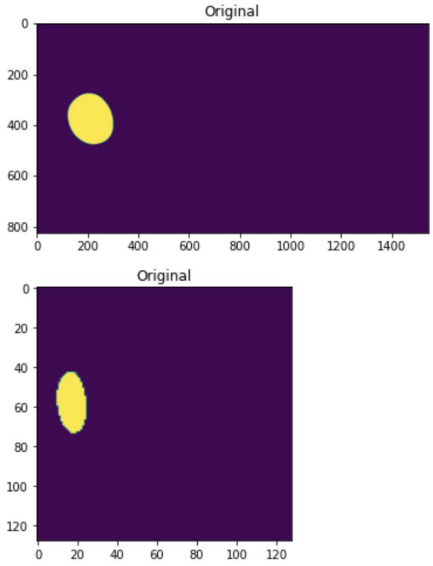


Resizing : Image resizing is needed to feed a smaller input image size rather than the original input size which would result in an increased number of parameters and a complex architecture. We resize our images to 128 x 128 pixels.

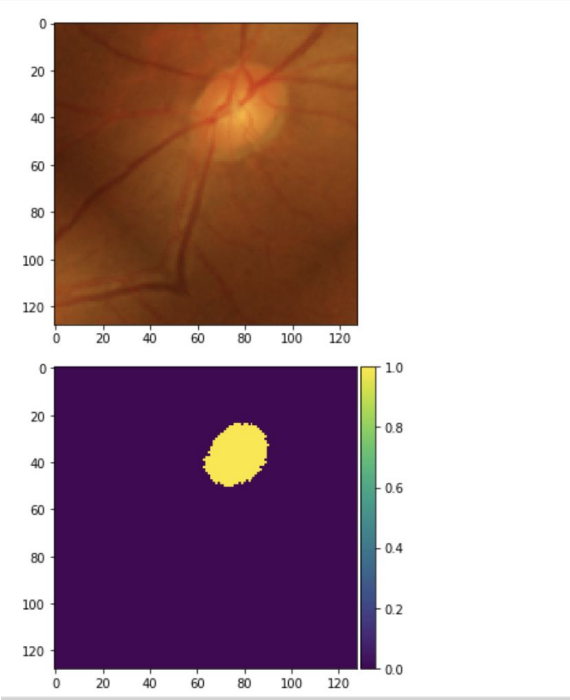
```
import matplotlib.image as mpimg
from keras.preprocessing.image import save_img
def processing(data):
    # loading image
    # Getting 100 images to work with
    img = [cv2.imread(i, cv2.IMREAD_UNCHANGED) for i in data]
    #print(img)
    try:
        print('Original size',img[0].shape)
    except AttributeError:
        print("shape not found")
    # ---------------------------------
    # setting dim of the resize
    height = 128
    width = 128
    dim = (width, height)
    res_img = []
    print(len(img))
    for i in range(len(img)):
        filename = os.path.basename(data[i])
        filename = os.path.splitext(filename)[0]
        res = cv2.resize(img[i], dim, interpolation=cv2.INTER_LINEAR)
        ## Scale of cv2 changes so converting back to RGB
        RGB_img = cv2.cvtColor(res, cv2.COLOR_BGR2RGB)
        img_array = img_to_array(RGB_img)
        save_img('/content/gdrive/My Drive/Drishti_RIM_Resized_Data/' + filename + '.png', img_array)
        #print("this is",img[i])
        #save_img('i', res)
        res_img.append(res)
    # Checcking the size
    try:
        print('RESIZED', res_img[1].shape)
    except AttributeError:
        print("shape not found")

    # Checcking the size
    print("RESIZED", res_img[1].shape)

    # Visualizing one of the images in the array
    original = res_img[1]
    print(type(original))

    display_one(img[1])
    display_one(original)
```

After resizing, the image mask looks as shown below :

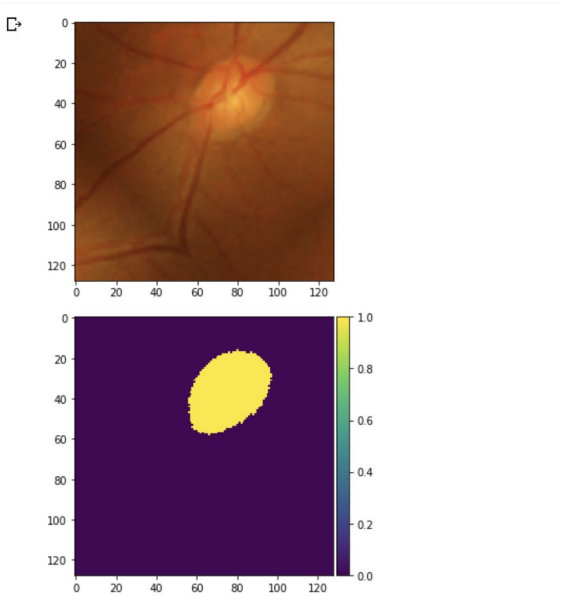
```
Original size (824, 1544, 3)
418
RESIZED (128, 128)
RESIZED (128, 128)
<class 'numpy.ndarray'>
```



And below figure shows a re-sized image and its corresponding cup mask.



Re-sized image and its corresponding disc mask for the same image:



2.1.4. Highlights of Modified U-Net Model

We discuss in a later section key modifications for our lighter U-Net model compared to deeper U-Net models. Our U-Net approach for fundus image segmentation includes two separate steps(two different models):

- Segmentation of optic cup
- Segmentation of optic disc

The U-Net segmentation model would be applied in separate processes to predict cup masks and disc masks respectively. The model to predict cup and mask has a same architecture, and uses same IOU and DICE coefficient for performance evaluation, but differs in a loss function.

```
Model: "model_8"
_________________________________________________________________________________________________
Layer (type)                 Output Shape              Param #     Connected to
=================================================================================================
input_9 (InputLayer)         [(None, 128, 128, 3)      0
_________________________________________________________________________________________________
lambda_8 (Lambda)            (None, 128, 128, 3)       0           input_9[0][0]
_________________________________________________________________________________________________
conv2d_164 (Conv2D)          (None, 128, 128, 16)      448         lambda_8[0][0]
_________________________________________________________________________________________________
batch_normalization_36 (BatchNo (None, 128, 128, 16)   48          conv2d_164[0][0]
_________________________________________________________________________________________________
dropout_78 (Dropout)         (None, 128, 128, 16)      0           batch_normalization_36[0][0]
_________________________________________________________________________________________________
conv2d_165 (Conv2D)          (None, 128, 128, 16)      2320        dropout_78[0][0]
_________________________________________________________________________________________________
batch_normalization_37 (BatchNo (None, 128, 128, 16)   48          conv2d_165[0][0]
_________________________________________________________________________________________________
max_pooling2d_35 (MaxPooling2D) (None, 64, 64, 16)     0           batch_normalization_37[0][0]
_________________________________________________________________________________________________
conv2d_166 (Conv2D)          (None, 64, 64, 32)        4640        max_pooling2d_35[0][0]
_________________________________________________________________________________________________
batch_normalization_38 (BatchNo (None, 64, 64, 32)     96          conv2d_166[0][0]
_________________________________________________________________________________________________
dropout_79 (Dropout)         (None, 64, 64, 32)        0           batch_normalization_38[0][0]
```

The last block of the model layer is comprised of three convolution layers.

```
conv2d_179 (Conv2D)             (None, 64, 64, 32)    9248        dropout_85[0][0]
_________________________________________________________________________________________________
batch_normalization_51 (BatchNo (None, 64, 64, 32)    96          conv2d_179[0][0]
_________________________________________________________________________________________________
conv2d_transpose_38 (Conv2DTran (None, 128, 128, 16) 2064         batch_normalization_51[0][0]
_________________________________________________________________________________________________
concatenate_38 (Concatenate)    (None, 128, 128, 32) 0           conv2d_transpose_38[0][0]
                                                                 batch_normalization_37[0][0]
_________________________________________________________________________________________________
conv2d_180 (Conv2D)             (None, 128, 128, 16) 4624        concatenate_38[0][0]
_________________________________________________________________________________________________
batch_normalization_52 (BatchNo (None, 128, 128, 16) 48          conv2d_180[0][0]
_________________________________________________________________________________________________
dropout_86 (Dropout)            (None, 128, 128, 16) 0           batch_normalization_52[0][0]
_________________________________________________________________________________________________
conv2d_181 (Conv2D)             (None, 128, 128, 16) 2320        dropout_86[0][0]
_________________________________________________________________________________________________
batch_normalization_53 (BatchNo (None, 128, 128, 16) 48          conv2d_181[0][0]
_________________________________________________________________________________________________
conv2d_182 (Conv2D)             (None, 128, 128, 1)  17          batch_normalization_53[0][0]
=================================================================================================
Total params: 1,945,521
Trainable params: 1,942,577
Non-trainable params: 2,944
```

Our initial training dataset for segmentation of cups includes 1189 images (RIM-ONE, DRISHTI, MESSIDOR, MAGRABIA, BINRUSHED & REFUGE image database). Our initial training dataset for segmentation of disc includes 704 images (RIM-ONE, DRISHTI, MAGRABIA, REFUGE image database). The remaining images were not included for disc segmentation because of the lower quality of disc masks from MESSIDOR and BINRUSHED.

The current lighter U-net model has 1,945,521 total parameters, which is meaningfully less than the more complex U-net we adopted earlier from Javier Civit and team's TPU-based UNET model.

We also used dropout after convolution layers to avoid overfitting, along with batch normalization layers.

We employed data augmentation techniques such as vertical and horizontal flips, rotations, zooms, and shifts. The code for data augmentation is shown here:

```
# Creating the training Image and Mask generator
image_datagen = image.ImageDataGenerator(shear_range=0.5, rotation_range=50, zoom_range=0.2, width_shift_range=0.2,
                                         height_shift_range=0.2, fill_mode='reflect')
mask_datagen = image.ImageDataGenerator(shear_range=0.5, rotation_range=50, zoom_range=0.2, width_shift_range=0.2,
                                        height_shift_range=0.2, fill_mode='reflect')
```

Our modified U-Net model was built with Tensorflow version ***2.2.0-rc3.*** We decided to upgrade our TensorFlow version from 1.0+ to the latest version of 2.2 due to these considerations:

- Some functions from TensorFlow version 1 had been deprecated in recent months with the TensorFlow community migrating to TensorFlow 2.0+;
- We had difficulty converting TensorFlow models generated by TensorFlow version 1 into the SavedModel object format required for TensorFlow Serving, a model server on GCP which we will briefly review in the last section of this report.

2.1.5. Segmentation Model Performance Evaluation Metrics

Segmentation problems often use specific loss functions such as Dice Loss and similarity metrics like Intersection-over-Union and Dice Coefficient. We referenced an article by Lars Nieradzik for performing different experiments comparing loss functions [4]. These metrics are similarity measures which calculate the degree of overlap between predicted masks and the corresponding ground truth masks. See section 3.2 for a review of image segmentation performance metrics.

2.1.6. U-Net Model Results - Disc and Cup segmentation

In this section we review performance differences from variations of U-Net models applied on the same datasets. These learnings on performance variances serve to help us better understand how modifications to a deep learning model's architecture, loss function differences, or other performance optimization modifications can impact our model's prediction performance.

2.1.6.1 Performance differences using different loss functions & scoring metric definitions

We performed model performance experiments by comparing test or validation set results by applying different loss functions. We summarize some learnings about the choice of loss functions: -

- Experimentation with different loss functions is often needed to learn which loss function(s) might provide better model performance for each specific case/problem;
- On how to choose between Dice/IoU versus cross-entropy loss functions: *"The main reason that people try to use dice coefficient or IoU directly is that the actual goal is maximization of those metrics, and cross-entropy is just a proxy which is easier to maximize using backpropagation."* [5]
- *"In addition, Dice coefficient performs better at class imbalanced problems by design."* [5] In our fundus image segmentation case, we believe because of the presence of pixel class imbalance (the mask output is a small fraction of the overall image) the Dice coefficient metric and Dice loss function are appropriate choices for our problem.
- The following discussion thread sheds light on why a combined "Dice coefficient plus Cross Entropy" loss function might outperform a single metric loss function - *"...each loss prioritizes different features in your image. It is up to you to decide which features are most important and then weigh the losses such that the outcome is acceptable. For example, dice loss puts more emphasis on imbalanced classes so if you weigh it more, your output will be more accurate/sensitive towards that goal. CE (cross-entropy) prioritizes the overall pixel-wise accuracy so some classes might suffer if they don't have enough representation to influence CE. Now when you add those two together you can play around with weighing the contributions of CE vs. Dice in your function such that the result is acceptable."* [6]

a.) **Binary Cross Entropy Loss , IOU (as defined above) & Accuracy**

We began model training using binary cross-entropy as our loss function, which attained a validation mean IOU score of 0.70 in 30 epochs.

```
Epoch 28/30
149/150 [============================>.] - ETA: 0s - loss: 0.0216 - iou_coef: 0.5106 - accuracy: 0.9886
Epoch 00028: val_loss improved from 0.02522 to 0.02493, saving model to UNet_Drishti_RIM_v2.h5
150/150 [==============================] - 8s 56ms/step - loss: 0.0215 - iou_coef: 0.5110 - accuracy: 0.9886 - val_loss: 0.0249 - val_iou_coef: 0.7211 - val_accuracy: 0.98
Epoch 29/30
150/150 [==============================] - ETA: 0s - loss: 0.0179 - iou_coef: 0.5620 - accuracy: 0.9899
Epoch 00029: val_loss did not improve from 0.02493
150/150 [==============================] - 8s 56ms/step - loss: 0.0179 - iou_coef: 0.5620 - accuracy: 0.9899 - val_loss: 0.0440 - val_iou_coef: 0.6669 - val_accuracy: 0.98
Epoch 30/30
150/150 [==============================] - ETA: 0s - loss: 0.0200 - iou_coef: 0.5418 - accuracy: 0.9892
Epoch 00030: val_loss did not improve from 0.02493
150/150 [==============================] - 8s 55ms/step - loss: 0.0200 - iou_coef: 0.5418 - accuracy: 0.9892 - val_loss: 0.0311 - val_iou_coef: 0.7001 - val_accuracy: 0.98
```

b.) Dice Coefficient, Dice Loss, IOU (as defined above) & Accuracy

To evaluate more performance metrics, we added the dice coefficient and changed our loss function to dice loss (as defined in the above section) from binary cross-entropy previously.

Below results show the last epochs after early stopping. Our validation IOU score is 0.62, and dice coefficient is 0.71.

```
Epoch 00005: val_loss improved from 0.27052 to 0.23346, saving model to UNet_Drishti_RIM_25_01.h5
200/200 [==============================] - 11s 55ms/step - loss: 0.3633 - iou_coef: 0.5048 - dice_coef: 0.6832 - accuracy: 0.9825 - val_loss: 0.2335 - val_iou_coef: 0.6390
Epoch 6/30
200/200 [==============================] - ETA: 0s - loss: 0.3525 - iou_coef: 0.5163 - dice_coef: 0.6956 - accuracy: 0.9829
Epoch 00006: val_loss did not improve from 0.23346
200/200 [==============================] - 11s 54ms/step - loss: 0.3525 - iou_coef: 0.5163 - dice_coef: 0.6956 - accuracy: 0.9829 - val_loss: 0.2563 - val_iou_coef: 0.6078
Epoch 7/30
200/200 [==============================] - ETA: 0s - loss: 0.3531 - iou_coef: 0.5143 - dice_coef: 0.6981 - accuracy: 0.9833
Epoch 00007: val_loss did not improve from 0.23346
200/200 [==============================] - 11s 54ms/step - loss: 0.3531 - iou_coef: 0.5143 - dice_coef: 0.6981 - accuracy: 0.9833 - val_loss: 0.3001 - val_iou_coef: 0.5692
Epoch 8/30
200/200 [==============================] - ETA: 0s - loss: 0.3344 - iou_coef: 0.5368 - dice_coef: 0.7155 - accuracy: 0.9844
Epoch 00008: val_loss did not improve from 0.23346
200/200 [==============================] - 11s 54ms/step - loss: 0.3344 - iou_coef: 0.5368 - dice_coef: 0.7155 - accuracy: 0.9844 - val_loss: 0.2441 - val_iou_coef: 0.6221
Epoch 00008: early stopping
```

c.) Dice Coefficient, Dice + Binary Cross Entropy Loss, IOU (as defined above) & Accuracy

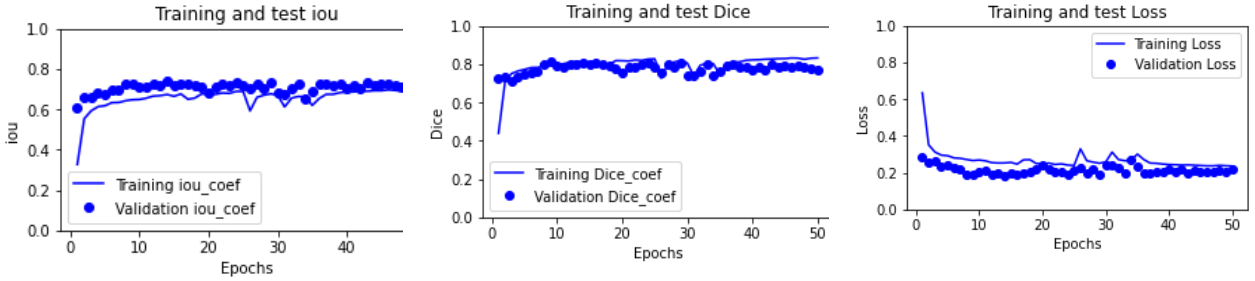
We further explored changing our model's loss function to "Dice score plus Binary Cross entropy", a hybrid loss function, which resulted in improvements in our performance metric results. We reached a validation IOU of 0.72, validation Dice score of 0.78 and validation loss of 0.19, higher than the results from models using other loss functions.

```
Epoch 00035: val_loss did not improve from 0.17290
200/200 [==============================] - 34s 170ms/step - loss: 0.2305 - iou_coef: 0.7031 - dice_coef: 0.8385 - accuracy: 0.9959 - val_loss: 0.1803 - val_iou_coef: 0.7459 - val_dice_coef: 0.8238 -
Epoch 36/40
200/200 [==============================] - ETA: 0s - loss: 0.2323 - iou_coef: 0.6987 - dice_coef: 0.8349 - accuracy: 0.9958
Epoch 00036: val_loss did not improve from 0.17290
200/200 [==============================] - 34s 168ms/step - loss: 0.2323 - iou_coef: 0.6987 - dice_coef: 0.8349 - accuracy: 0.9958 - val_loss: 0.1827 - val_iou_coef: 0.7406 - val_dice_coef: 0.8136 -
Epoch 37/40
200/200 [==============================] - ETA: 0s - loss: 0.2410 - iou_coef: 0.6918 - dice_coef: 0.8298 - accuracy: 0.9958
Epoch 00037: val_loss did not improve from 0.17290
200/200 [==============================] - 34s 170ms/step - loss: 0.2410 - iou_coef: 0.6918 - dice_coef: 0.8298 - accuracy: 0.9958 - val_loss: 0.1960 - val_iou_coef: 0.7282 - val_dice_coef: 0.7959 -
Epoch 38/40
200/200 [==============================] - ETA: 0s - loss: 0.2336 - iou_coef: 0.7015 - dice_coef: 0.8376 - accuracy: 0.9959
Epoch 00038: val_loss did not improve from 0.17290
200/200 [==============================] - 34s 170ms/step - loss: 0.2336 - iou_coef: 0.7015 - dice_coef: 0.8376 - accuracy: 0.9959 - val_loss: 0.2003 - val_iou_coef: 0.7270 - val_dice_coef: 0.7910 -
Epoch 39/40
200/200 [==============================] - ETA: 0s - loss: 0.2330 - iou_coef: 0.7024 - dice_coef: 0.8388 - accuracy: 0.9959
Epoch 00039: val_loss did not improve from 0.17290
200/200 [==============================] - 34s 169ms/step - loss: 0.2330 - iou_coef: 0.7024 - dice_coef: 0.8388 - accuracy: 0.9959 - val_loss: 0.2051 - val_iou_coef: 0.7219 - val_dice_coef: 0.7977 -
Epoch 40/40
200/200 [==============================] - ETA: 0s - loss: 0.2306 - iou_coef: 0.7022 - dice_coef: 0.8387 - accuracy: 0.9959
Epoch 00040: val_loss did not improve from 0.17290
200/200 [==============================] - 34s 169ms/step - loss: 0.2306 - iou_coef: 0.7022 - dice_coef: 0.8387 - accuracy: 0.9959 - val_loss: 0.1935 - val_iou_coef: 0.7338 - val_dice_coef: 0.7991 -
```

2.1.6.2 U-Net Model Results (Segmentation of Cup)

For our final U-Net cup segmentation, we used Dice Coefficient, IOU and *Dice + Binary Cross Entropy Loss* as the performance metrics.

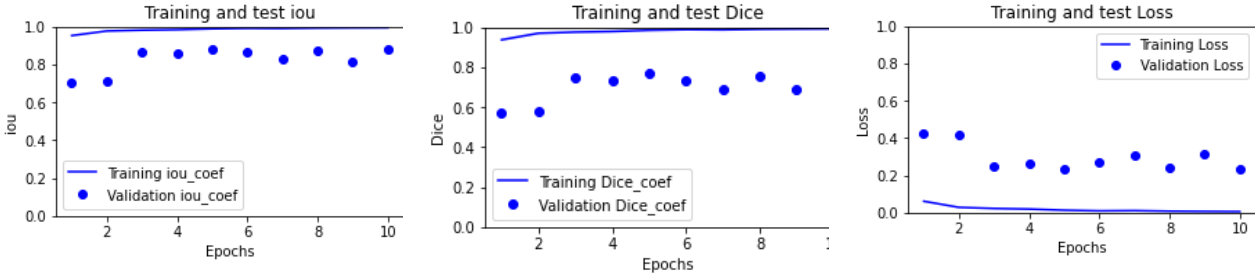
Below figures show our model's training and test metrics.



2.1.6.3 U-Net Model Results (Segmentation of Disc)

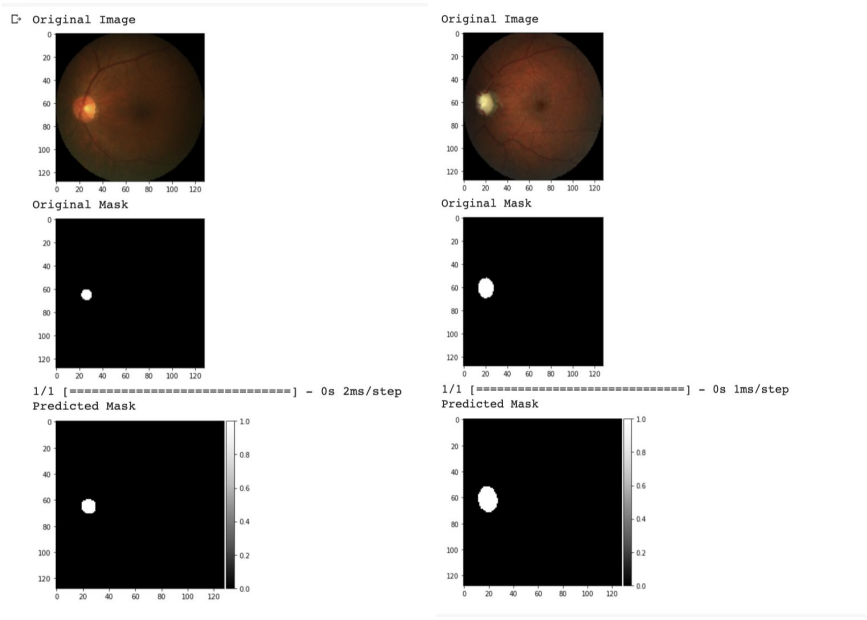
For disc segmentation, we performed the same experiments as in cup segmentation. We found that *Dice Coefficient Loss*, Dice Coefficient and IOU metrics to be the metrics that performed well for our light U-Net model and gave us the best results. For disc segmentation performance, we achieved a validation IOU score of 0.88 and validation Dice coefficient of 0.76. The final validation loss was 0.23.

```
Epoch 5/40
250/250 [==============================] - ETA: 0s - loss: 0.0131 - iou_coef: 0.9898 - dice_coef: 0.9869 - accuracy: 0.9778
Epoch 00005: val_loss improved from 0.25050 to 0.23198, saving model to /content/gdrive/My Drive/Drishti_RIM_REFUGE_discs_14_v30_200.h5
250/250 [==============================] - 36s 142ms/step - loss: 0.0131 - iou_coef: 0.9898 - dice_coef: 0.9869 - accuracy: 0.9778 - val_loss: 0.2320 - val_iou_coef: 0.8834 - val_dice_coef: 0.7680
Epoch 6/40
250/250 [==============================] - ETA: 0s - loss: 0.0097 - iou_coef: 0.9923 - dice_coef: 0.9903 - accuracy: 0.9823
Epoch 00006: val_loss did not improve from 0.23198
250/250 [==============================] - 35s 138ms/step - loss: 0.0097 - iou_coef: 0.9923 - dice_coef: 0.9903 - accuracy: 0.9823 - val_loss: 0.2687 - val_iou_coef: 0.8705 - val_dice_coef: 0.7313
Epoch 7/40
250/250 [==============================] - ETA: 0s - loss: 0.0109 - iou_coef: 0.9916 - dice_coef: 0.9891 - accuracy: 0.9810
Epoch 00007: val_loss did not improve from 0.23198
250/250 [==============================] - 35s 139ms/step - loss: 0.0109 - iou_coef: 0.9916 - dice_coef: 0.9891 - accuracy: 0.9810 - val_loss: 0.3093 - val_iou_coef: 0.8331 - val_dice_coef: 0.6907
Epoch 8/40
250/250 [==============================] - ETA: 0s - loss: 0.0079 - iou_coef: 0.9937 - dice_coef: 0.9921 - accuracy: 0.9849
Epoch 00008: val_loss did not improve from 0.23198
250/250 [==============================] - 34s 137ms/step - loss: 0.0079 - iou_coef: 0.9937 - dice_coef: 0.9921 - accuracy: 0.9849 - val_loss: 0.2446 - val_iou_coef: 0.8742 - val_dice_coef: 0.7554
Epoch 9/40
250/250 [==============================] - ETA: 0s - loss: 0.0066 - iou_coef: 0.9948 - dice_coef: 0.9934 - accuracy: 0.9869
Epoch 00009: val_loss did not improve from 0.23198
250/250 [==============================] - 34s 137ms/step - loss: 0.0066 - iou_coef: 0.9948 - dice_coef: 0.9934 - accuracy: 0.9869 - val_loss: 0.3116 - val_iou_coef: 0.8181 - val_dice_coef: 0.6884
Epoch 10/40
250/250 [==============================] - ETA: 0s - loss: 0.0057 - iou_coef: 0.9954 - dice_coef: 0.9943 - accuracy: 0.9881
Epoch 00010: val_loss did not improve from 0.23198
250/250 [==============================] - 35s 138ms/step - loss: 0.0057 - iou_coef: 0.9954 - dice_coef: 0.9943 - accuracy: 0.9881 - val_loss: 0.2323 - val_iou_coef: 0.8811 - val_dice_coef: 0.7677
Epoch 00010: early stopping
```



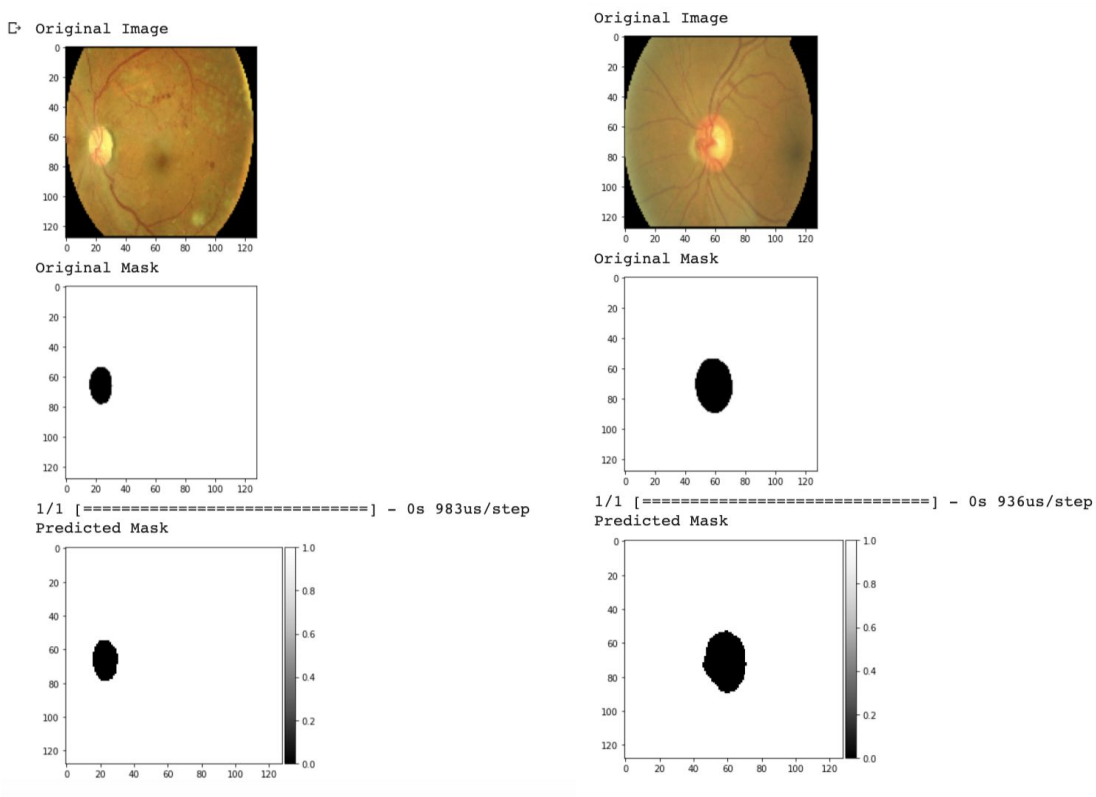
2.1.6.3 U-Net Model Predictions (Segmentation of Cup and Disc)

Below we can see the cup predictions of a Normal And Glaucoma Image



Normal Image **Glaucoma Image**

Below we can see the disc predictions, from some of the test images.



2.1.7 U-Net model - experimenting with a lighter U-Net

In the earlier part of our project, the first U-Net segmentation model we explored was a more complex U-net referencing the work from Javier Civit and team's TPU-based UNET model [7].

Civit team's U-Net model adapted the work based on Sevastopolsky and team [1]. Civit adapted their model to be more compatible to run on TPU servers, along with a number of modifications such as adding more data augmentation steps with modified brightness. The Civit model was applied to two fundus image datasets - DRISHTI and RIM-ONE (which we continue to use in our subsequent modified U-Net models) and was built with TensorFlow version 1.3, while we upgraded our TensorFlow library to version 2.2 for this project.

2.1.7.1 Why consider experimenting with a lighter deep learning model?

Below are some considerations in favor of a lighter neural network:

- Training time required for more complex deep learning models using only GPU servers could take a considerable amount of time, reaching day(s).
- Server resource constraints / access to TPU servers: State-of-the-art segmentation methods that rely on very deep networks are not always easy to train without very large training datasets and tend to be relatively slow to run on standard GPUs, without access to TPU servers.

- Recent advances in U-Net model research are making progress on improving speed and performance for resource-constrained U-Nets. For example, W. Wang, K. Yu, J. Hugonot, P. Fua and M. Salzmann introduces in their paper "Recurrent U-Net for Resource-Constrained Segmentation" [8] a novel recurrent U-Net architecture that preserves the compactness of the original U-Net, while substantially increasing its performance to the point where it outperforms the state of the art on several benchmarks. The authors mention in their paper: *"state-of-the-art semantic segmentation techniques...rely on very deep networks, which makes them ill-suited in resource-constrained scenarios, such as real-time applications and when there are only limited amounts of training data. In such cases, more compact networks are preferable."* [8]

Main modifications for our lighter U-Net model include:

- We experimented with a similar architecture by trimming the number of Convolution layers from over 20 layers to 9 layers;
- The lighter U-Net architecture resulted in a considerably lower number of feature parameters at 1,941,715, compared to the 5.6 million parameters of the more complex TPU-based UNET model from Civet et al. [7];
- The model training time required for the light U-Net model was consequently much less than that for the deeper U-Net, which required up to day(s) of training time on a GPU.

Please refer to the Appendix section for a visual graph representation of the U-Net model from Civit & team and for our lighter U-Net model architecture.

2.1.7.2 Performance differences between a deeper U-Net and a lighter U-Net

Our original objective was to observe and understand model performance differences between "deep", many layered U-Net (adoption of Civet et al.'s TPU-based U-Net [7]) and a lighter U-Net with less convolution layers on the same image datasets.

	Cup Segmentation prediction: Highest validation scores achieved by model version	Loss function used	Intersection -over-Union (IOU) metric	Dice Coef metric	Pixel Accuracy metric
1	(Deep U-Net) Civet et al. TPU-based U-Net: enhanced data augmentation on ***2 datasets*** (replaced original with Dice loss function)	Dice loss	0.65	0.81	0.90
2	Light U-Net v.1: ***2 datasets*** with Dice loss function	Dice loss	0.58	0.74	0.88
3	Light U-Net v.2: ***4 datasets*** with Dice + Binary Cross Entropy Loss function	Dice loss + Binary Cross Entropy	0.65	0.78	0.99
4	Light U-Net v.2: ***6 datasets*** with Dice + Binary Cross Entropy Loss function	Dice loss + Binary Cross Entropy	0.72	0.78	NA

We summarize our learnings on prediction performance comparing test data (or validation data) prediction performance of various versions of deep versus light U-Net models.

- Both deep and lighter versions of U-Net models perform reasonably well measured by Dice coefficient. Our lighter U-Net models saw test data performance of 0.74 to 0.78 on Dice coefficient, compared to 0.81 for a deeper U-Net from Civet et al..

- We realize image segmentation prediction often involves the issue of imbalanced classes (having a much higher proportion of pixels in the image belonging to the positive class) and thus Accuracy as a prediction performance metric is not a very robust measure of segmentation prediction performance, which is why we evaluate our segmentation models on IOU score and Dice score, rather than pixel Accuracy.

- Our light U-Net models' training and test performance results are quite close, which gives us some confidence that our models are not overfitting.

- For our second light U-Net model, model performance improved further after we added two additional fundus datasets to our training data to arrive at a total of 694 original images.

- Adding two more datasets from RIGA and REFUGE databases further improved the performance of our cup segmentation performance, such that our highest performing U-Net model is not too far from the base-line deeper U-Net on the Dice coefficient and outperformed the deeper U-Net on the IoU score (model 4 IoU of 0.72 vs. model 1 IoU of 0.65). Our last model was trained on a total of 1189 original images.

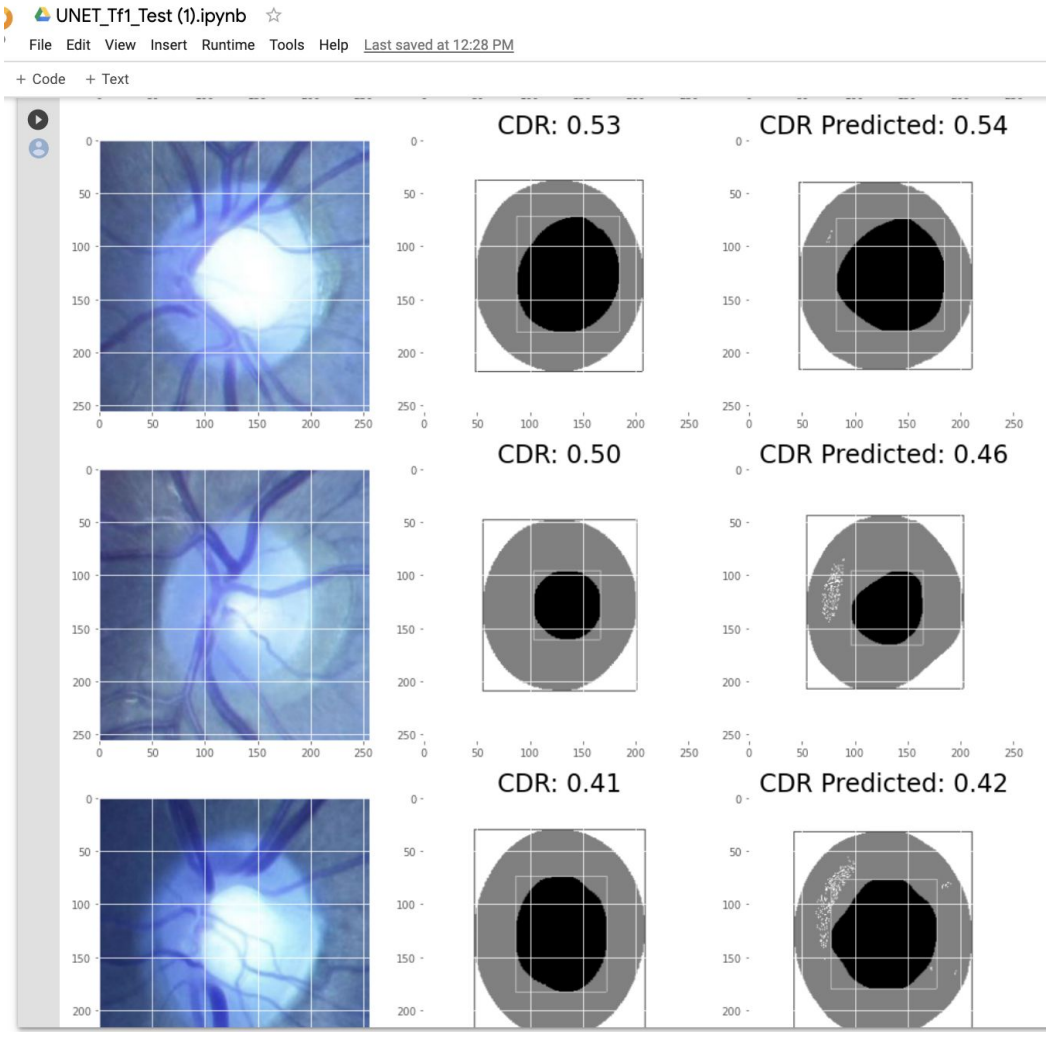
2.1.7.3 Areas for improvement on U-Nets for future consideration

We highlight areas for improvement and further exploration based on an examination of our model results from above:

a) Despite our light U-Net models achieving decent overall Dice and IoU scores on test/validation data in the 0.70-0.8 range, some portion of our U-NET models' predicted output masks did not perform well upon visual inspection of the predicted mask compared against the ground truth mask. It appears to us there remains a performance evaluation disconnect between performance levels suggested by degree-of-overlap segmentation measures (referring to Dice & IoU scores) and a visual examination of how reasonable our predicted masks "visually match" ground truth masks -- the accuracy of such an attempt is also difficult to quantify.
b) We observed in our GCP-implemented U-Net model work that U-Net mask output results vastly improved in cases where fundus images were successfully cropped, i.e. cases when the optic disc center is correctly located and the original image is successfully cropped as part of image pre-processing. With this observation, we hypothesize that an improvement in image cropping (such that the image is zoomed in to focus on our desired area-of-interest) is an important factor in improving subsequent model training and hence prediction performance. In section 2.2.4 we discuss how better disc cropping can help save the model from learning useless features, which can in turn improve model performance.
c) Different loss functions which rely on a combination of loss metrics, for instance, dice loss combined with binary cross entropy, or other customized loss functions, should be explored further as a method to improve model performance.
d) We also realize there might be a need for using different loss functions for segmentation of cup and the segmentation of disc.
e) Toward the final weeks of our project, we began to explore a complementary and possibly more robust way to evaluate fundus segmentation prediction performance, that is to compare the extracted CDR ratio from predicted segmentation masks, against the

extracted CDR ratio from ground truth masks. See below snapshot for examples of ground truth CDRs placed next to predicted CDRs (for ease of visual inspection).

In one of the later versions of a deep U-Net model (more convolution layers added), after experimenting with a number of loss functions and modifying the cropping function, we saw marked improvement on the basis of predicted CDR vs. ground truth CDR, to the range of over 70% of CDR predictions being within 0.1 of the ground truth CDR. Perhaps further work can be done on incorporating predicted vs. ground truth CDR as a model performance evaluation metric or as part of a loss function.



2.1.8 U-Net model - Data and Code

The dataset specific for this model is uploaded on the GCP bucket in cds_2020_dataset/UNetLightModelData

The code which runs both cup and disc models is uploaded on GCP bucket in cds_2020_code_repository/UNetLightModel.

VM cdr-unet-wc-1 on GCP has full running applications for preprocessing, generating images and files for bounding boxes for AutoML models as well as modules for running and testing U-Net application.

2.2. Object Detection with GCP AutoML Vision

2.2.1. Objectives

Object detection models have two objectives: classification and localization.

Classification tries to correctly label an object, for example a couch, chair, dog, cat, etc. Our object detection model is trained with two labels, Cup and Disc.

Localization distinguishes between multiple objects in an image and provides the position of the object within the image as well as the rectangular bounds for the region containing the object. Evaluation involves a comparison of ground truth and predicted bounding boxes.

Predictions take an image and for each detected object returns the label, a segmentation mask with bounding box coordinates, and a confidence score. The confidence score is the confidence of the bounding box compared with all other predicted bounding boxes for the same object in the dataset. In the case of a live camera feed the dataset comprises multiple frames that are buffered and sent to the model. The confidence score allows you to rank the results.

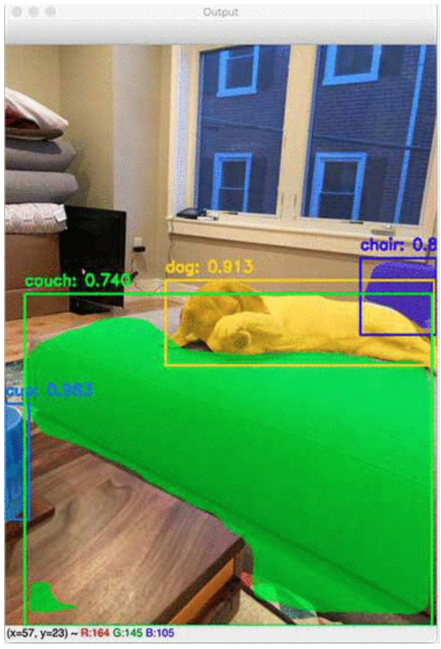
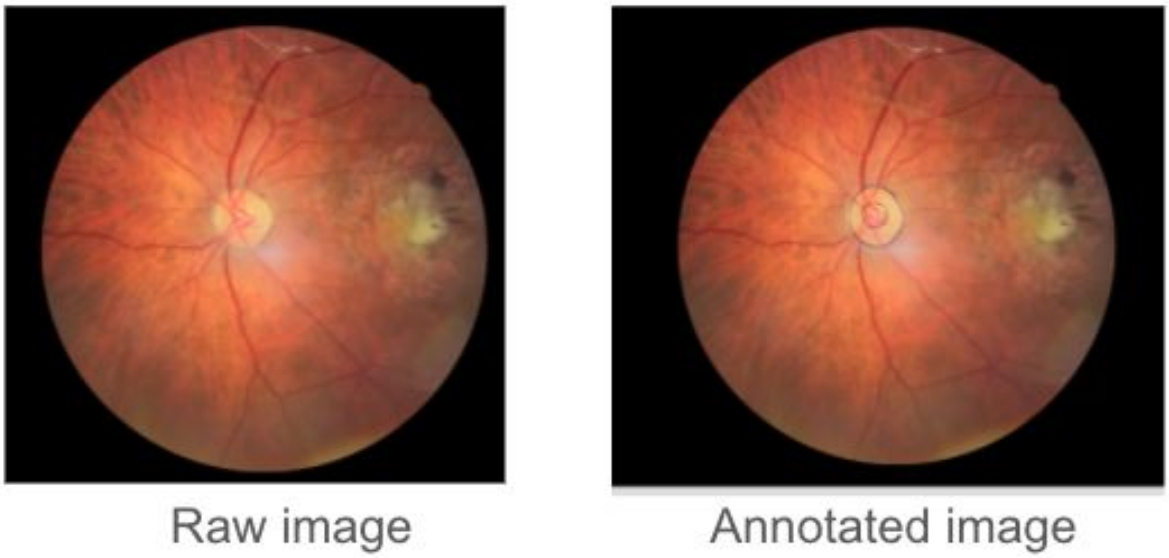


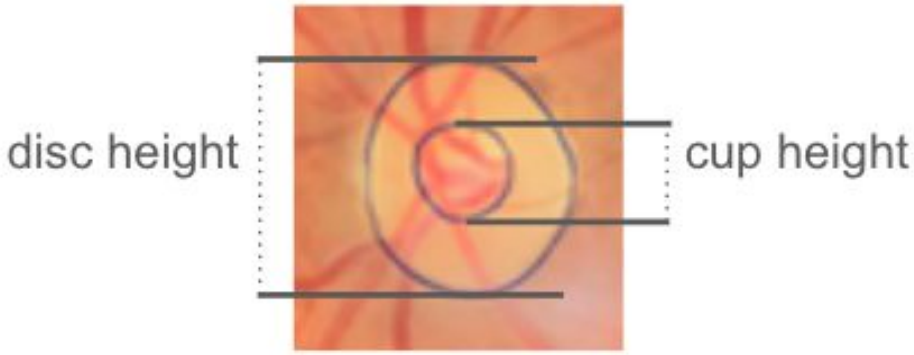
Figure 2.2.1. Output of a MaskRCNN object detection model trained on the Microsoft COCO (common objects in context) dataset.

Training uses the image features, ground truth labels and ground truth bounding boxes or segmentation masks.

Model performance is evaluated on both classification and localization. The concept of intersection over union (IoU) is used to compare the ground truth and predicted bounding box. An IoU of 1 is a perfect overlap of the predicted and ground truth bounding boxes.

For this project we would like to train a custom object detection model that can take a fundus image, classify the cup and the disc and return accurate bounding boxes for each. Accurate bounding boxes will allow us to return a CDR prediction.





CDR = Cup Height / Disc Height

2.2.2. Object Detection with AutoML Vision

AutoML Vision allows you to train custom machine learning models for object detection. An API and user interface are available for importing and labeling data, viewing images and labels, training and reviewing model evaluation metrics, deploying in the cloud and running predictions in the cloud. In addition to cloud deployment, AutoML can train lighter models for deployment to a smartphone or microcontroller.

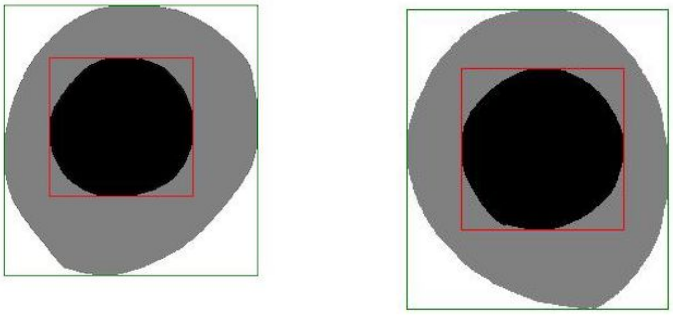
Model Input

AutoML takes the location of the image on cloud storage, the ground truth label, and the ground truth bounding box coordinates. Notice that bounding box coordinates for each label are fed in separately.

TRAIN	gs://cdr_images_bounding_boxes/img/BinRushed/BinRushed1/image10prime.jpg	Cup	0.41667	0.41919	0.48443	0.41919	0.48443	0.509	0.41667	0.509
TRAIN	gs://cdr_images_bounding_boxes/img/BinRushed/BinRushed1/image10prime.jpg	Disc	0.39478	0.38826	0.49747	0.38826	0.49747	0.544	0.39478	0.544
TEST	gs://cdr_images_bounding_boxes/img/BinRushed/BinRushed1/image11prime.jpg	Cup	0.52273	0.43939	0.58838	0.43939	0.58838	0.522	0.52273	0.522
TEST	gs://cdr_images_bounding_boxes/img/BinRushed/BinRushed1/image11prime.jpg	Disc	0.5101	0.41162	0.60774	0.41162	0.60774	0.552	0.5101	0.552

Before we can train the AutoML object detection model we must first derive ground truth bounding boxes from the annotated images.

2.2.3. Preprocessing



First segmentation masks are created from the annotated images. From the segmentation masks object localization information is created, giving us the ground truth bounding box coordinates to provide as input to AutoML.

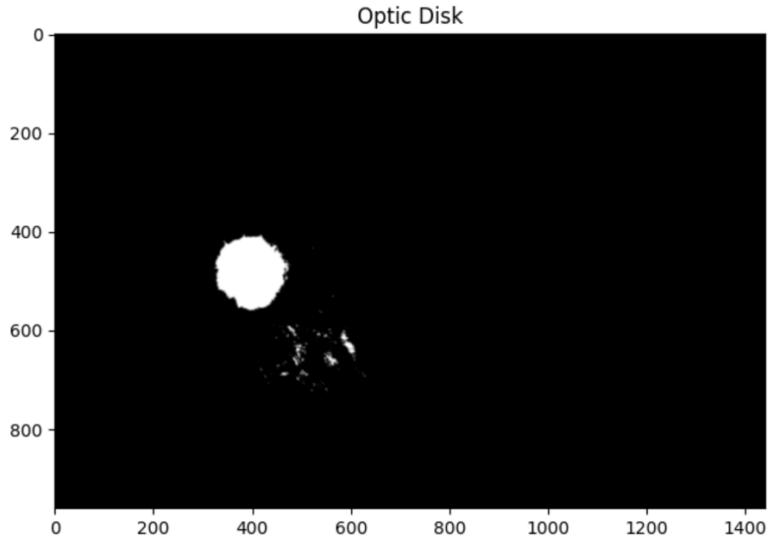
VALIDATE	gs://cdr_images_bounding_boxes/img/REFUGE/Train/Glaucoma/g0013.jpg	Disc	0.05932	0.41780	0.18644	0.41780	0.18644	0.553	0.05932	0.553
VALIDATE	gs://cdr_images_bounding_boxes/img/REFUGE/Train/Glaucoma/g0013.jpg	Cup	0.08145	0.44115	0.15725	0.44115	0.15725	0.518	0.08145	0.518
TRAIN	gs://cdr_images_bounding_boxes/img/REFUGE/Train/Glaucoma/g0007.jpg	Disc	0.07062	0.41294	0.21846	0.41294	0.21846	0.579	0.07062	0.579
TRAIN	gs://cdr_images_bounding_boxes/img/REFUGE/Train/Glaucoma/g0007.jpg	Cup	0.09322	0.43337	0.21751	0.43337	0.21751	0.562	0.09322	0.562
TRAIN	gs://cdr_images_bounding_boxes/img/REFUGE/Train/Glaucoma/g0006.jpg	Disc	0.06638	0.41780	0.20245	0.41780	0.20245	0.585	0.06638	0.585
TRAIN	gs://cdr_images_bounding_boxes/img/REFUGE/Train/Glaucoma/g0006.jpg	Cup	0.10546	0.44212	0.19727	0.44212	0.19727	0.561	0.10546	0.561
TRAIN	gs://cdr_images_bounding_boxes/img/REFUGE/Train/Glaucoma/g0012.jpg	Disc	0.09557	0.40856	0.21563	0.40856	0.21563	0.552	0.09557	0.552
TRAIN	gs://cdr_images_bounding_boxes/img/REFUGE/Train/Glaucoma/g0012.jpg	Cup	0.12618	0.44650	0.19586	0.44650	0.19586	0.518	0.12618	0.518
TRAIN	gs://cdr_images_bounding_boxes/img/REFUGE/Train/Glaucoma/g0004.jpg	Disc	0.10217	0.39348	0.22034	0.39348	0.22034	0.519	0.10217	0.519
TRAIN	gs://cdr_images_bounding_boxes/img/REFUGE/Train/Glaucoma/g0004.jpg	Cup	0.12476	0.41634	0.20292	0.41634	0.20292	0.500	0.12476	0.500
TEST	gs://cdr_images_bounding_boxes/img/REFUGE/Train/Glaucoma/g0010.jpg	Disc	0.15348	0.38035	0.28766	0.38035	0.28766	0.533	0.15348	0.533
TEST	gs://cdr_images_bounding_boxes/img/REFUGE/Train/Glaucoma/g0010.jpg	Cup	0.19068	0.40224	0.28578	0.40224	0.28578	0.498	0.19068	0.498

These bounding boxes will also allow us to generate a cropped image which is zoomed on the center of the disc.

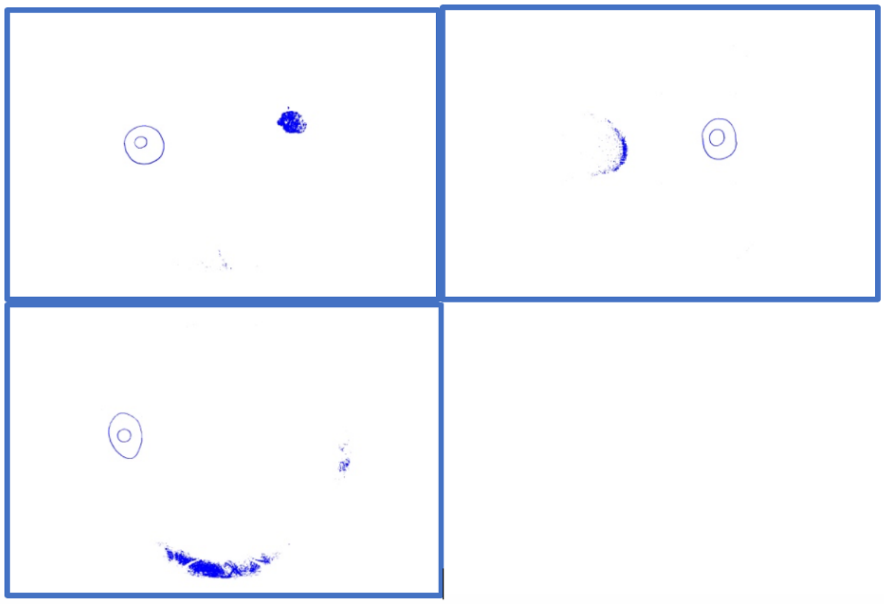
Masking

For various datasets in the RIGA collection, masks were extracted based on graders' annotations which are the manual markings around the cup and the disc. Some of the challenges were poor image quality, poor quality markings in a blue color that was indistinguishable at pixel level from the background, equally lit or dark areas in the image, and low contrast. A step by step approach was developed to create masks out of these images.

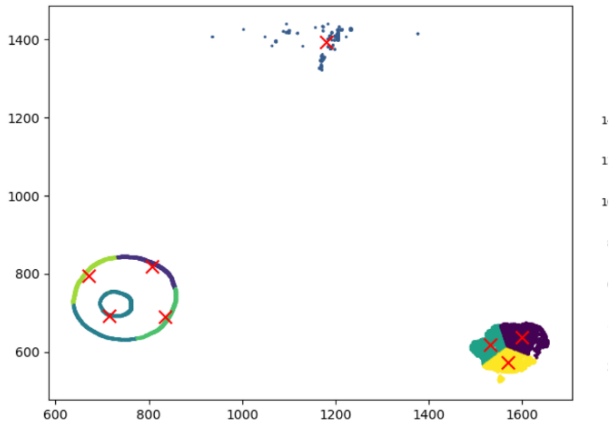
Step 1—Narrow down Disc area

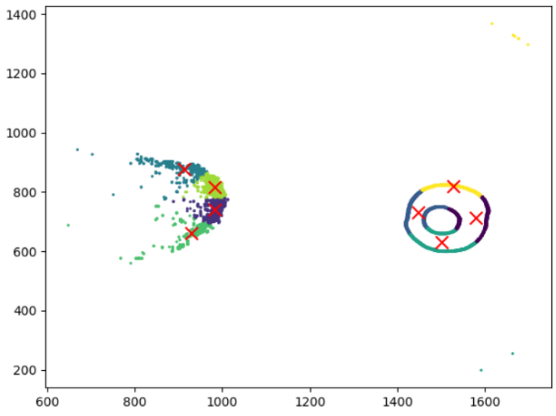


Step 2—Identify Disc Areas

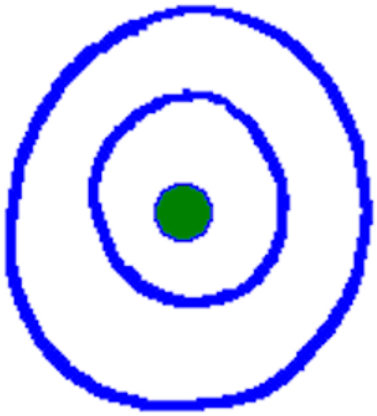


Step 3: Identify and isolate Clusters

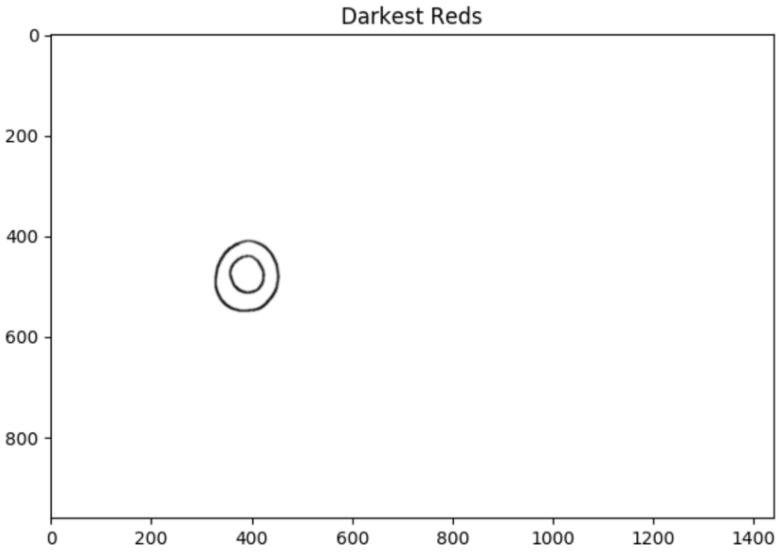




Step 3—Outlier removed -- Disc Center Identified



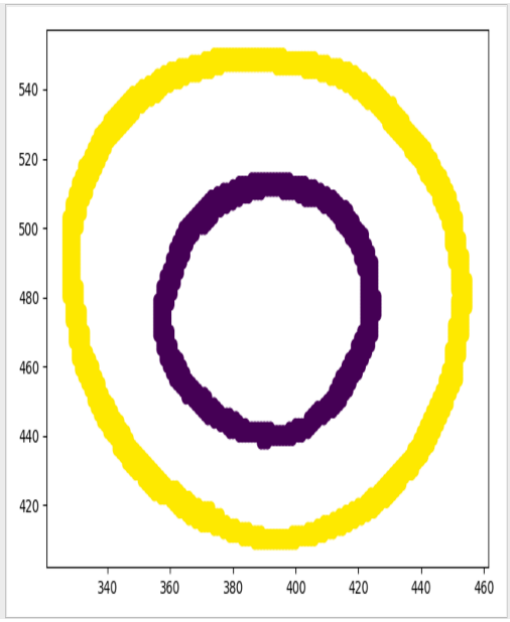
Step 4—Identify Disc and Cup outlines

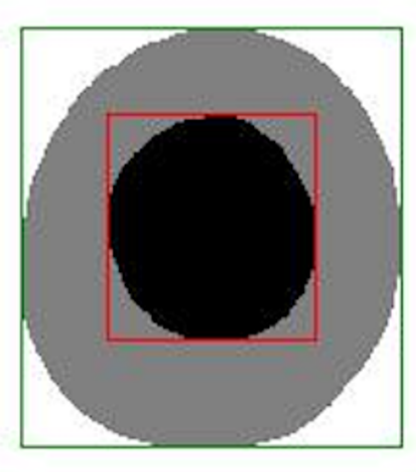


Step 5 — Separate into Disc and Cup:

Now we are left with pixel points of the marking. However, we still need to identify from this list of points which belong to the inner and which are part of the outer shape. There are some

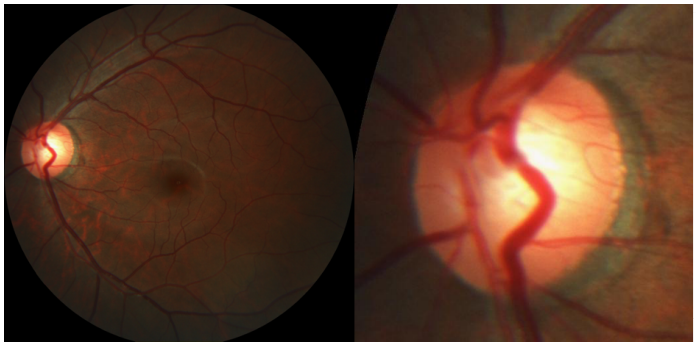
techniques, like drawing lines to infinity and note their points of intersection to identify the contained closed shape within an outer closed shape but that were found to be too inefficient. We used a littler known extension of K-means, called Spectral clustering to identify the cup and disc.

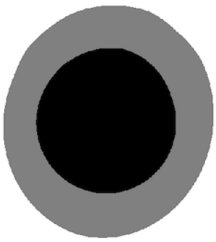




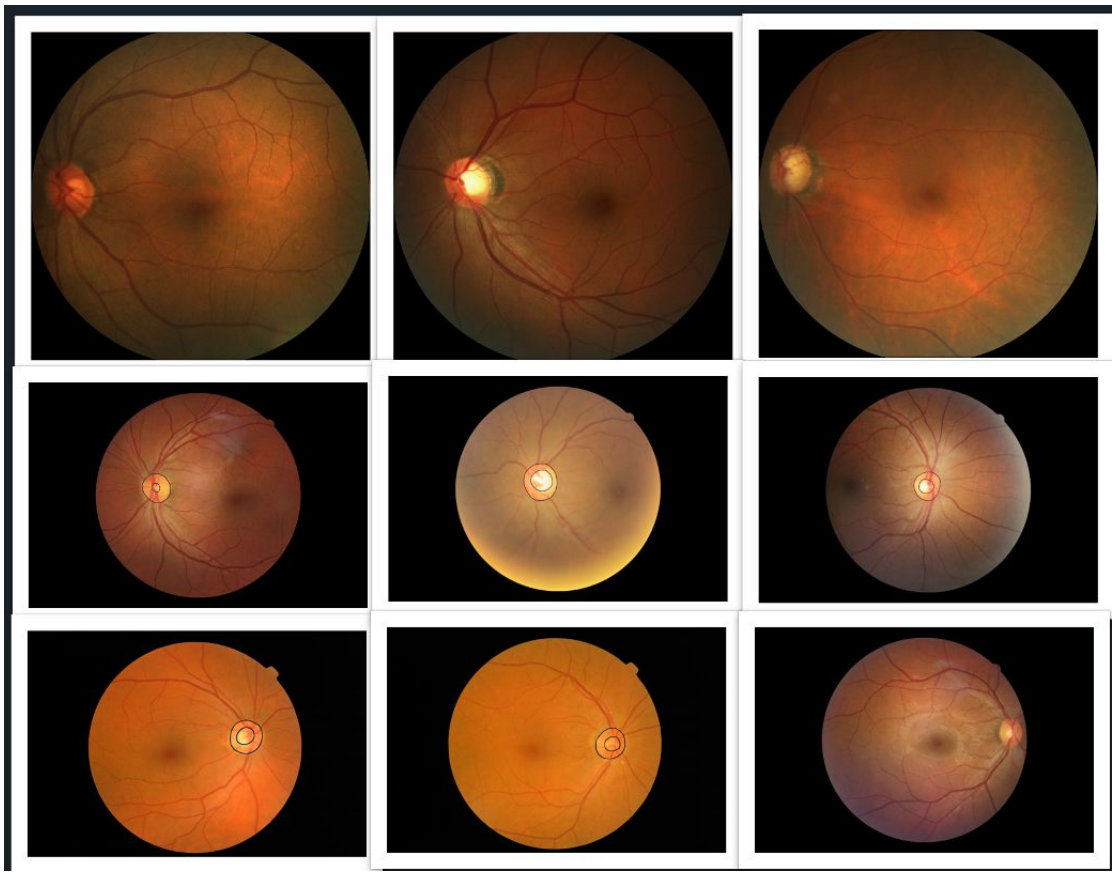
2.2.4. Disc Cropping

Disc cropping was required to prevent the system from learning useless features, which greatly improved performance and accuracy.





Disc cropping is challenging for many reasons including the presence of equally bright areas in the image as well as the location of the disc which can be anywhere in the image, as seen below. Some discs are found to be at extreme left, others are located on right while some are in the center.



2.2.5. A novel technique to identify the Disc Area of Interest

Step 1- Create a Template

For All Images in the Dataset:

1. Extract (380x380) optic disk
2. Extract the channels: RGB and bin them into histograms.
3. Store Min, Max, Median Values

Take the average of all histograms.

Calculate aggregate statistics.

For any new image:

Based on Stats, if point can lie on the disc:

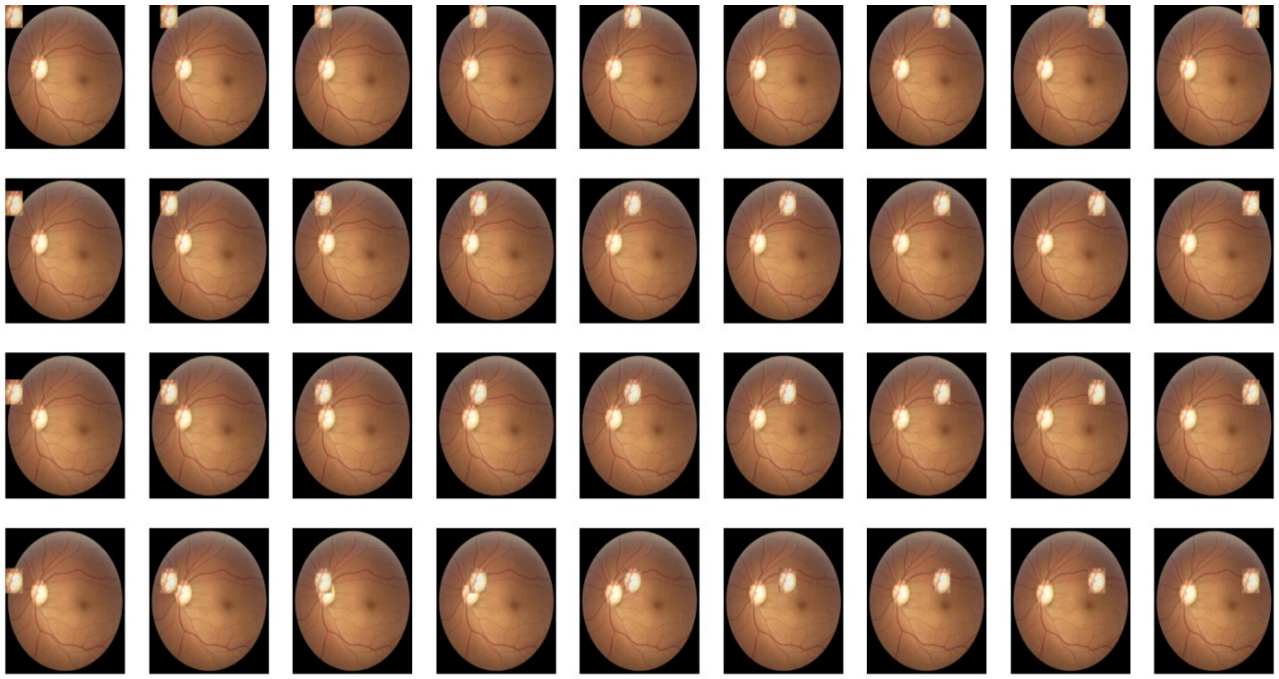
1. Create a (380x380) sliding window.
2. Extract the color components for the window and store their histograms.
3. Find c for each channel:

$$c = \frac{1}{\left(1 + \Sigma_i (a_i - b_i)^2\right)}$$

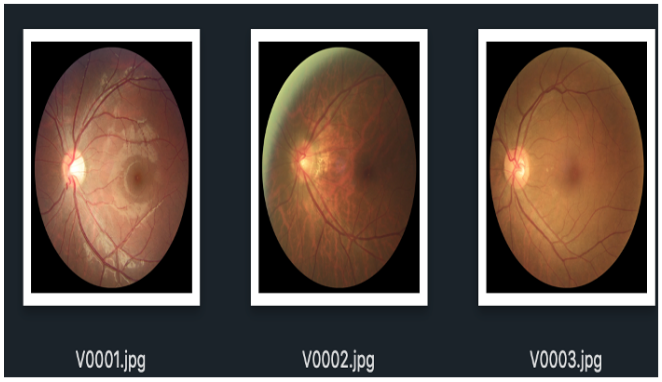
4. Find combined c:

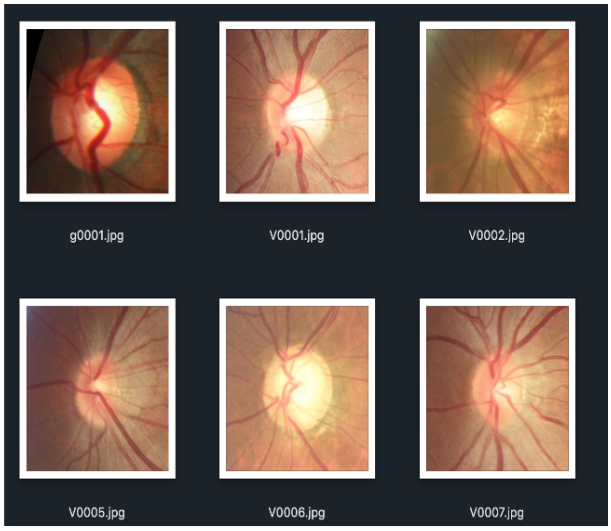
$$c(i,j) = t_r \times c_r + t_g \times c_g \times t_b \times c_b$$

5. Max value of c is the center of the disc.



Results

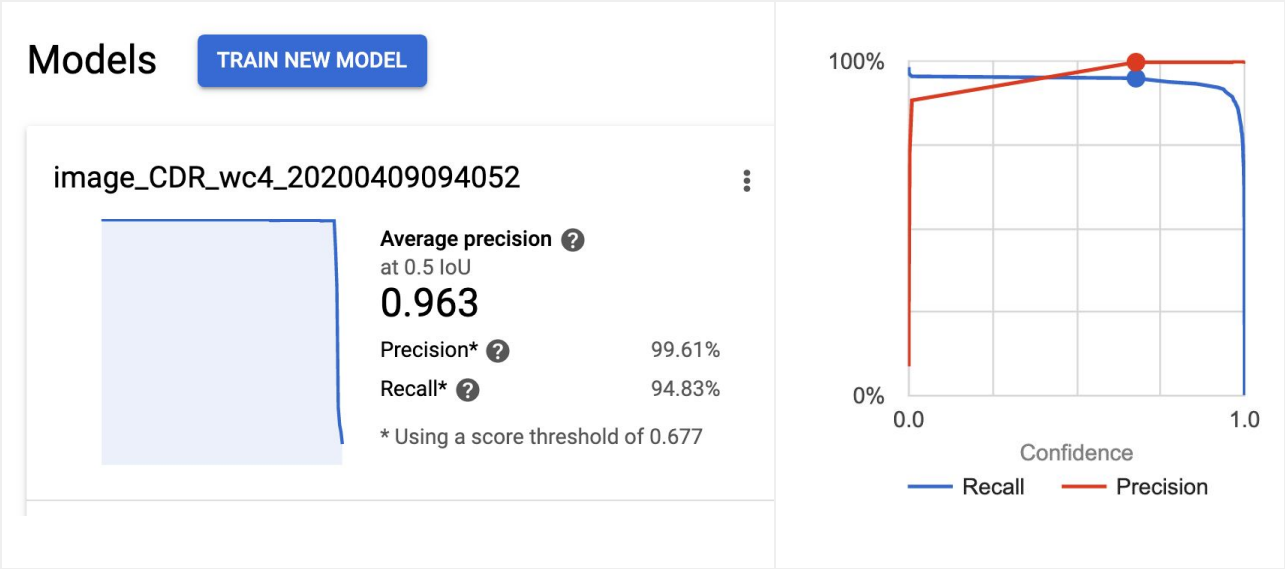


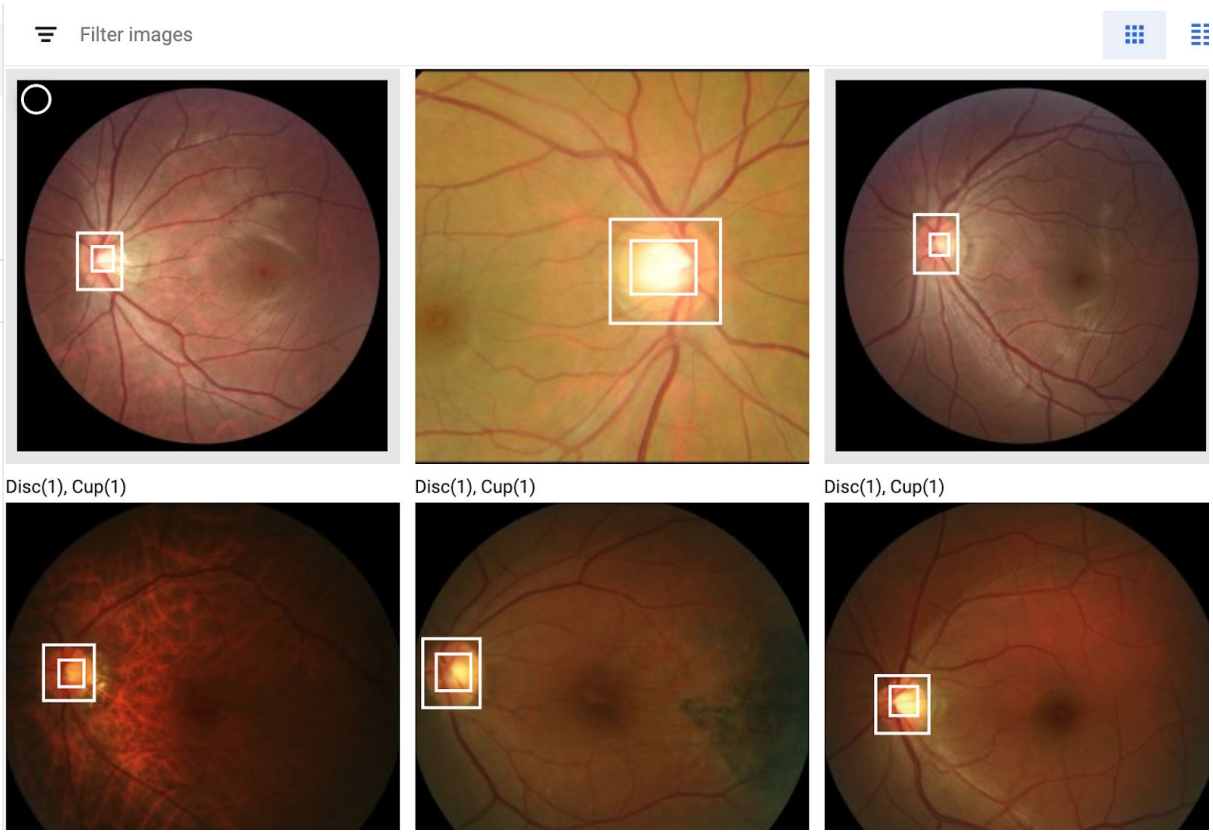


2.2.5. Ground Truth Analysis and Model Training

Training was done after generating masks and bounding boxes for about 4700 images, in which 1500 images were unique images while the rest were markings of the same image by different graders. In the RIGA dataset, the disc and cup in every image was annotated (drawn) by six different clinicians which we will call graders. After running a competition round between each

grader, where accuracy was noted, the grader with the best result was selected. This resulted in our best performing object detection model.





2.2.6. Image Selection during Model Training

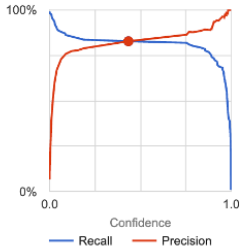
One question was whether to select a grader or to employ a blended approach. Six models were created, one for each grader. Each was given images from a different dataset that it had not seen. Final scores were noted and the the best or mean of two was employed to train our model:

Grader Assessments (1-2)

All labels

Total images	160
Test items	57
Total objects	116
Object to image avg	2.04
Precision	82.76%
Recall	82.76%

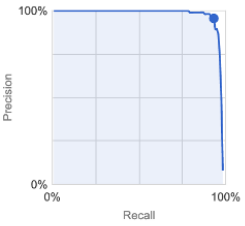
Use the slider to see which confidence threshold works best for your model on the precision-recall tradeoff curve.
Learn more about these metrics and graphs.

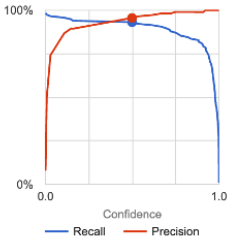


All labels

Total images	160
Test items	57
Total objects	116
Object to image avg	2.04
Precision	95.58%
Recall	93.1%

Use the slider to see which confidence threshold works best for your model on the precision-recall tradeoff curve.
Learn more about these metrics and graphs.



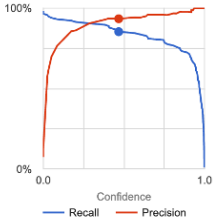


Grader Assessments (3-4)

All labels

Total images	147
Test items	56
Total objects	116
Object to image avg	2.07
Precision	93.4%
Recall	85.34%

Use the slider to see which confidence threshold works best for your model on the precision-recall tradeoff curve.
Learn more about these metrics and graphs.



All labels

Total images	160
Test items	57
Total objects	116
Object to image avg	2.04
Precision	98.13%
Recall	90.52%

Use the slider to see which confidence threshold works best for your model on the precision-recall tradeoff curve.
Learn more about these metrics and graphs.

Grader Assessments (5-6)

All labels

Total images	160
Test items	57
Total objects	116
Object to image avg	2.04
Precision	92.92%
Recall	90.52%

Use the slider to see which confidence threshold works best for your model on the precision-recall tradeoff curve.
Learn more about these metrics and graphs.

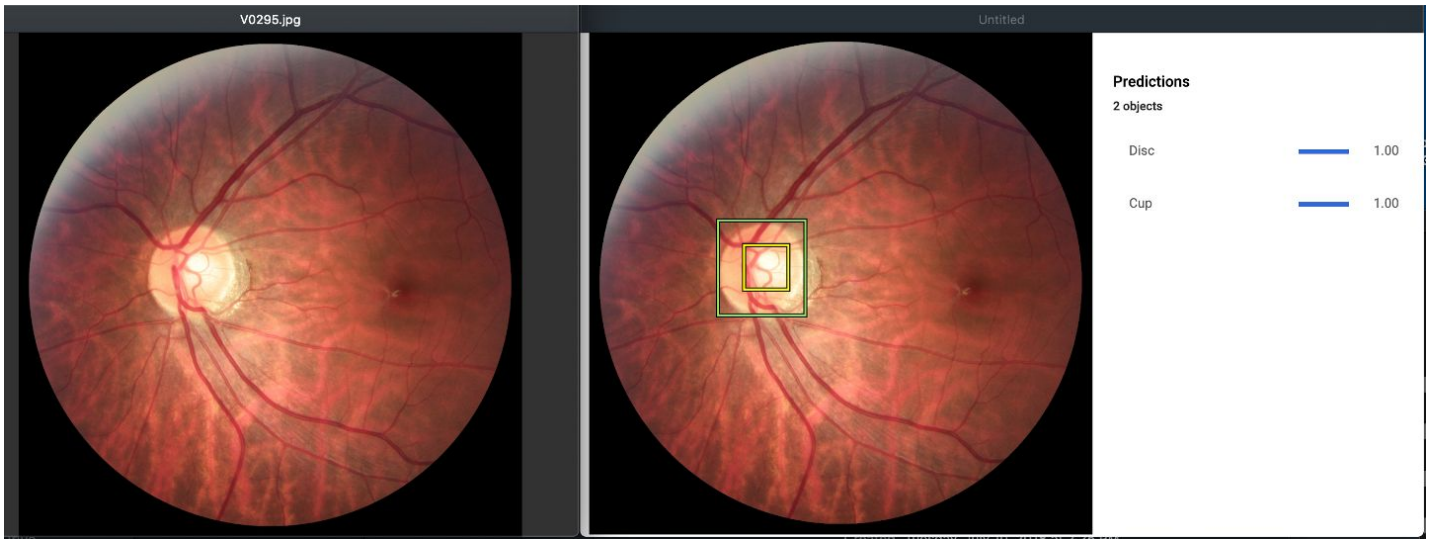
All labels

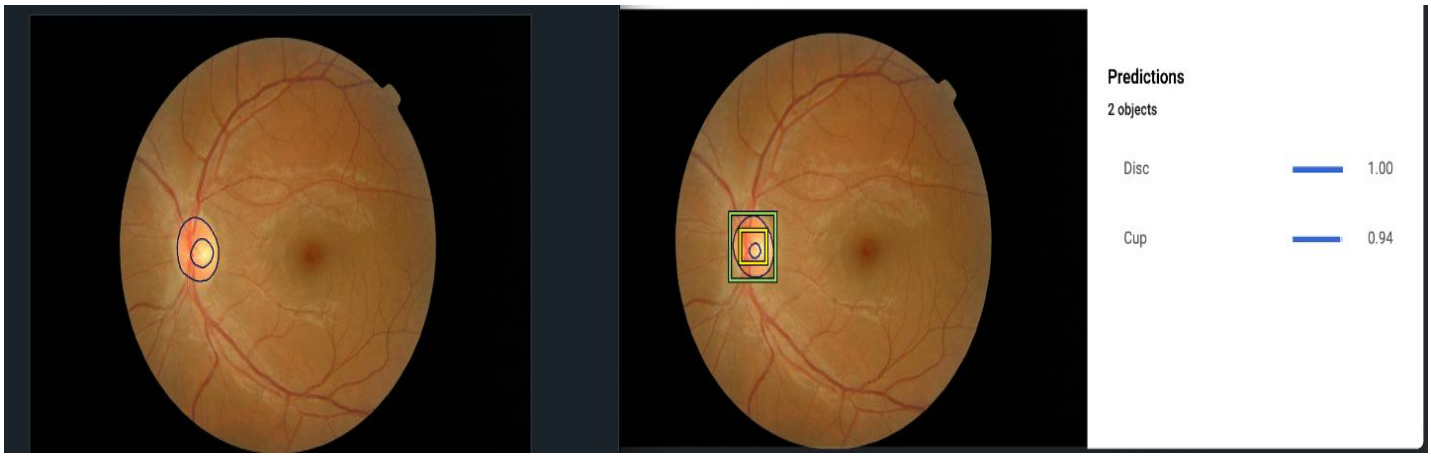
Total images	160
Test items	57
Total objects	116
Object to image avg	2.04
Precision	95.58%
Recall	93.1%

Use the slider to see which confidence threshold works best for your model on the precision-recall tradeoff curve.
Learn more about these metrics and graphs.

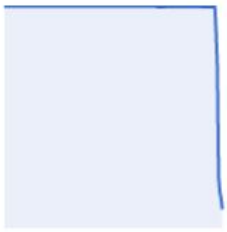
2.2.7. Results

2.2.7.1 Online Predictions - Cloud Deployment





2.2.7.2. Cloud Model Evaluation



Average precision
at 0.5 IoU
0.963

Precision*	99.61%
Recall*	94.83%

* Using a score threshold of 0.677

Model ID	IOD5622423077376753664
Created	Apr 9, 2020, 9:41:04 PM
Data	1,070 images
Model type	Cloud High Accuracy

Cup

Total images	811
Test items	259
Total objects	271
Object to image avg	1.05
Precision	99.22%
Recall	94.1%

Disc

Total images	811
Test items	259
Total objects	271
Object to image avg	1.05
Precision	100%
Recall	94.83%

True positives

Your model correctly predicted **Disc** on these images

Predicted Ground Truth

Score: 0.99905246

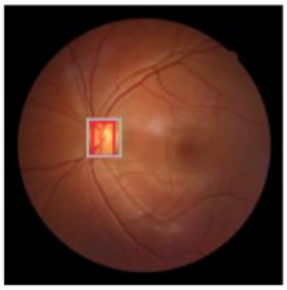
Score: 0.9990706

False negatives

Your model failed to predict an accurate bounding box for **Disc** on these images

Predicted Ground Truth

Score(s): 0.000184912



Score(s): 0.000001996

2.2.7.3 Mobile Model Evaluation

Average precision
at 0.5 IoU
0.937

Precision*	84.87%
Recall*	93.17%

* Using a score threshold of 0.488

Model ID	IOD2762074364043067392
Created	Apr 14, 2020, 3:33:16 PM
Data	1,070 images
Model type	Mobile High Accuracy

Cup

Total images	811
Test items	259
Total objects	271
Object to image avg	1.05
Precision	99.22%
Recall	94.1%

Disc

Total images	811
Test items	259
Total objects	271
Object to image avg	1.05
Precision	100%
Recall	94.83%

2.3. Image Classification with GCP AutoML Vision

2.3.1. Objective

Image classification models try to assign one or more labels to an image. Unlike object detection there is no localization objective.

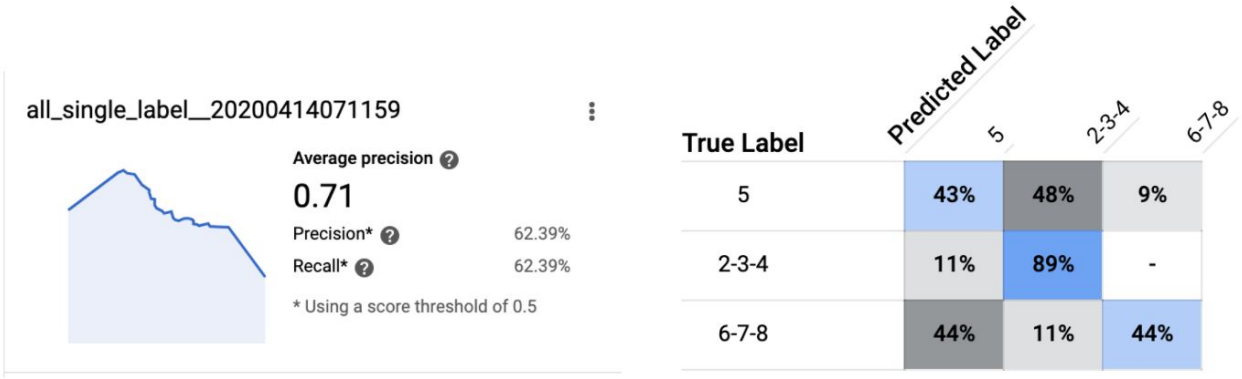
The objective of our image classification model is to directly predict the CDR for a fundus image. In a clinical setting the cup to disc ratio is expressed as a fraction to one decimal point, e.g. 0.3. For image classification purposes we can consider each possible CDR as a label.

Predictions take an image and return the predicted label(s) along with a confidence score. A confidence score is the confidence of the label compared with all other predicted labels for the same object in the dataset.

2.3.2. Models

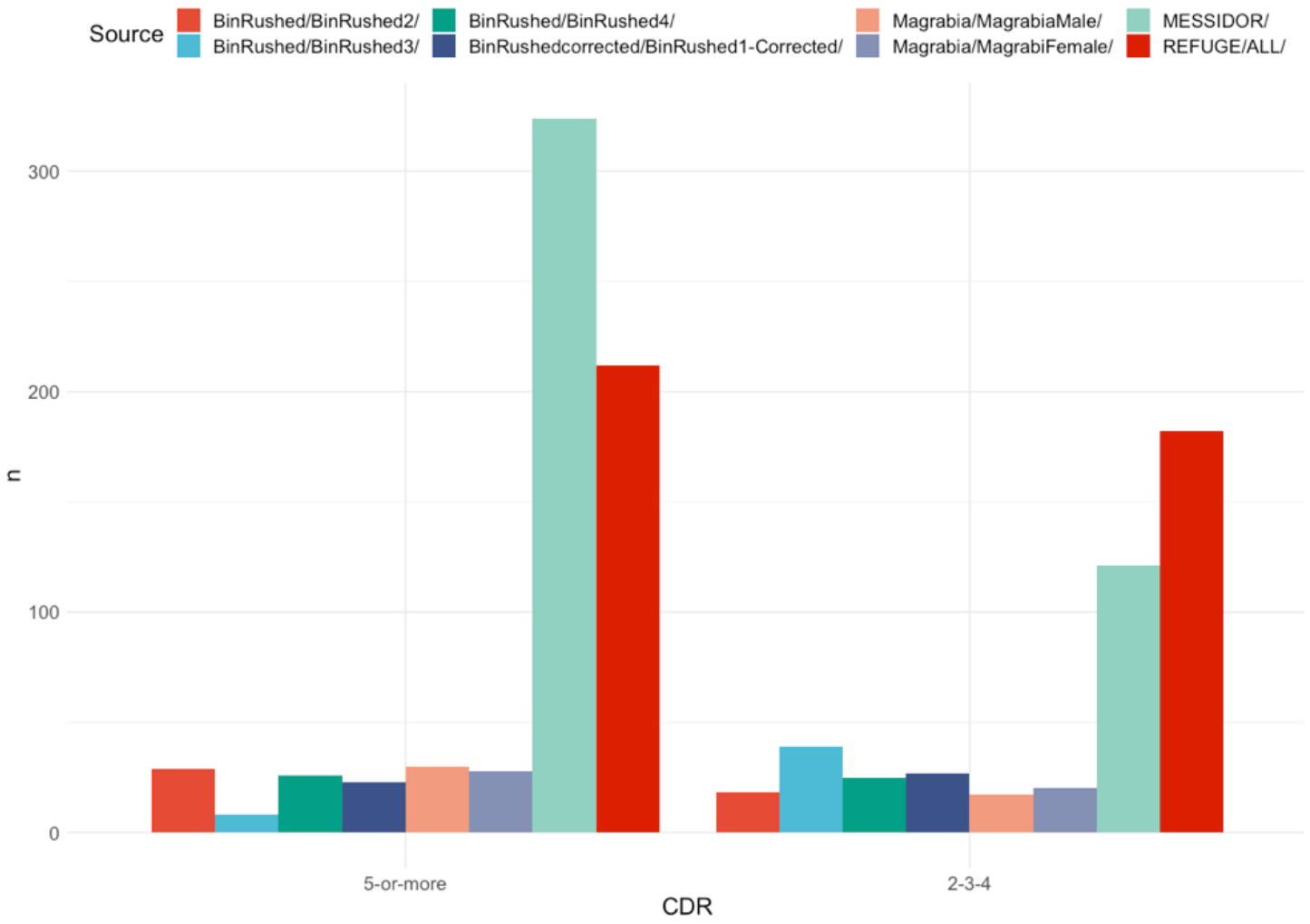
The cup and disc bounding boxes that we extracted from the RIGA and REFUGE images were used to derive ground truth CDRs for image classification.

Initial attempts to classify individual labels, e.g. 0.2, 0.3, ..., 0.8 performed poorly with average precision around 50%. Binning the CDR improved performance somewhat.



RIGA and REFUGE datasets with three labels

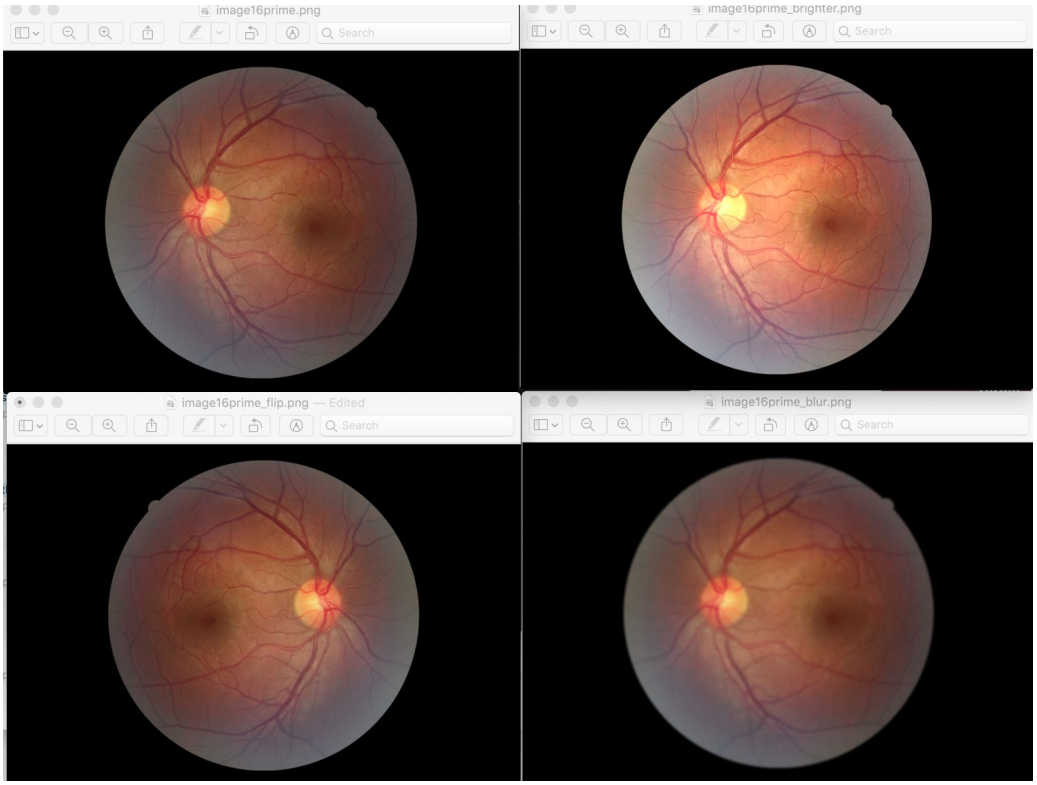
Better performance was achieved with two CDR ranges (1) 2-3-4 and (2) 5-or-more and by augmenting the dataset. The chart below shows how the original fundus images from RIGA and REFUGE are distributed between the two labels and by source.


Source
BinRushed/BinRushed2/
BinRushed/BinRushed3/
BinRushed/BinRushed4/
BinRushedcorrected/BinRushed1-Corrected/
Magrabia/MagrabiaMale/
Magrabia/MagrabiFemale/
MESSIDOR/
REFUGE/ALL/
300
200
100
0
n
5-or-more
2-3-4
CDR

2.3.3. Data Augmentation Methods

Using the ground truth bounding boxes that we previously derived, we cropped the images around the disc and cup area and added these cropped images to our training data.

We also experimented with the following augmentation methods.



Clockwise from upper left: original image, brightened image, with Gaussian blur applied, flipped on the vertical axis

In order to include augmented images in the Training set, we first split the dataset into TRAIN, VALIDATION and TEST and only included augmented images when the original image was assigned to training. We set the ground truth CDR for each augmented image to that of the original image.

TRAIN	gs://cds-2020-dataset-us-central/BinRushed/BinRushed2/image16prime.png	2-3-4
TRAIN	gs://cds-2020-dataset-us-central/BinRushed/BinRushed2/image16prime_brighter.png	2-3-4
TRAIN	gs://cds-2020-dataset-us-central/BinRushed/BinRushed2/image16prime_crop.png	2-3-4

TRAIN	gs://cds-2020-dataset-us-central/BinRushed/BinRushed2/image16prime_darker.png	2-3-4
TRAIN	gs://cds-2020-dataset-us-central/BinRushed/BinRushed2/image16prime_flip.png	2-3-4

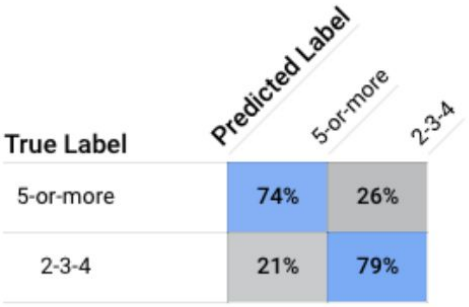
Augmenting the dataset using crops, brighter, darker and vertical flipping improved performance. Adding a Gaussian blur negatively impacted performance and was not used in the final version of this model.

2.3.4. Results

2.3.4.1. Cloud Model Evaluation

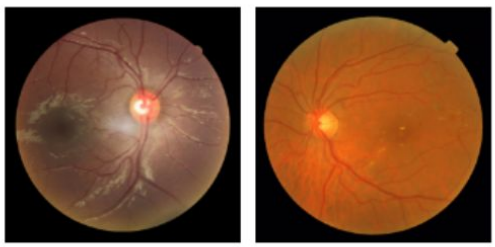
Average precision
0.878
Precision* 75.86%
Recall* 75.86%
* Using a score threshold of 0.5

Model ID	ICN7383154609818173440
Created	Apr 26, 2020, 3:44:11 PM
Base model	ICN8207876291580395520
Data	5,339 images
Model type	Cloud



True positives

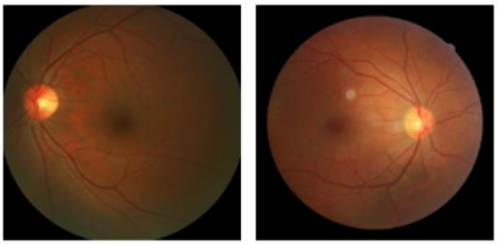
Your model correctly predicted **5-or-more** on these images



Score: 0.9493024 Score: 0.9531048

False negatives

Your model should have predicted **5-or-more** on these images



Score(s): 0.13857701 Score(s): 0.13266793

2.3.4.2. Mobile Model Evaluation

Average precision

0.802

Precision*	70.69%
Recall*	70.69%

* Using a score threshold of 0.5

Model ID	ICN7038066288370909184
Created	Apr 26, 2020, 1:42:14 PM
Base model	ICN6404184640818511872
Data	5,339 images
Model type	Mobile High Accuracy

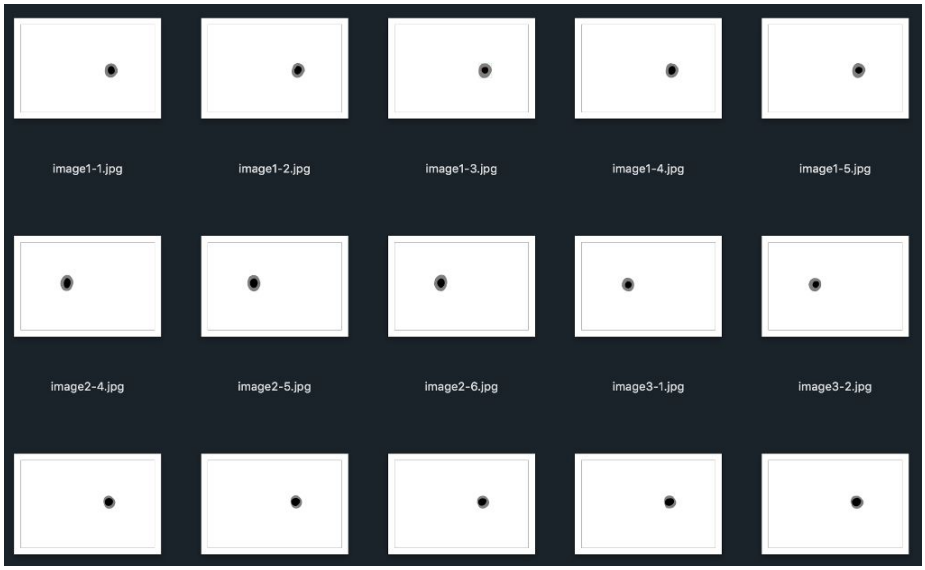
True Label \ Predicted Label	5-or-more	2-3-4
5-or-more	79%	21%
2-3-4	42%	58%

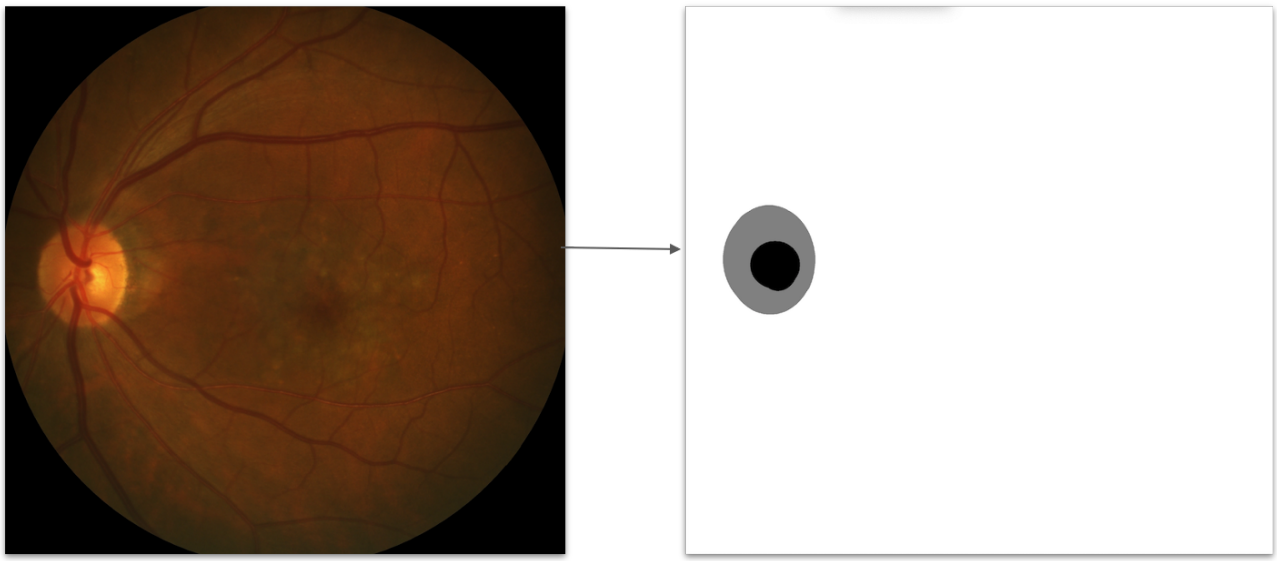
2.4. GCP UNet Segmentation model

This implementation based on UNet architecture uses RFEUGE and RIGA datasets. It uses the generated masks and works off cropped images, which are correctly recognized after first locating the center of the disc.

Masks:

As a first step masks of various sizes were created.





Evaluation:

It was evaluated in 4 different UNet models of various depths and complexities, 4 optimizers and 10 loss functions.
In order to address class imbalance, it was seen that standard loss functions returned misleading numbers for accuracy. Custom versions of loss functions were developed for Dice, Pixel loss, and Binary cross entropy.

2.4.1 Loss functions

2.4.2 Custom Pixel loss:

Function is geared towards rewarding matches around the disc area.

```
def custom_pixel_loss(y_true,y_pred):
    mask_val = 1
    indices = tf.where(tf.equal(y_true, mask_val))
    total = tf.cast(tf.count_nonzero(tf.equal(y_true, mask_val)), tf.float32)
    y_true_mask = tf.gather_nd(y_true, indices)
    y_pred_mask = tf.gather_nd(y_pred, indices)
    y_true_f = tf.keras.backend.flatten(y_true_mask)
    y_pred_f = tf.keras.backend.flatten(y_pred_mask)
    total_pred = tf.cast(tf.count_nonzero(tf.equal(y_pred, mask_val)), tf.float32)
    TP = tf.cast(tf.keras.backend.sum(y_true_f * y_pred_f), tf.float32)
    FP= tf.cast(tf.cond(tf.greater(total_pred,TP), lambda: total_pred-TP, lambda: 0.0),tf.float32)
    FN = tf.cast(tf.cond(tf.greater(total,TP), lambda: total-TP, lambda: 0.0),tf.float32)
    loss = 1/(1+(total-TP)+(0.1* FP) + (0.1*FN))
    return loss
```

2.4.3 Weighted Dice:

```
def weighted_dice_coef_1(y_true, y_pred):
    mean = 0.95
    w_1 = 1 / mean ** 2
    w_0 = 1 / (1 - mean) ** 2
    y_true_f_1 = tf.keras.backend.flatten(y_true)
    y_pred_f_1 = tf.keras.backend.flatten(y_pred)
    y_true_f_0 = tf.keras.backend.flatten(1 - y_true)
    y_pred_f_0 = tf.keras.backend.flatten(1 - y_pred)

    intersection_0 = tf.keras.backend.sum(y_true_f_0 * y_pred_f_0)
    intersection_1 = tf.keras.backend.sum(y_true_f_1 * y_pred_f_1)

    return 2 * (w_0 * intersection_0 + w_1 * intersection_1) / (
                (w_0 * (tf.keras.backend.sum(y_true_f_0) + tf.keras.backend.sum(y_pred_f_0))) + (
                    w_1 * (tf.keras.backend.sum(y_true_f_1) + tf.keras.backend.sum(y_pred_f_1))))
```

2.4.4 Pixel loss weighted:

```
def pixel_wise_loss_2(y_true, y_pred):
    pos_weight = tf.constant([[90, 10.0]])
    loss = tf.nn.weighted_cross_entropy_with_logits(
        y_true,
        y_pred,
        pos_weight,
        name=None
    )

    return tf.keras.backend.mean(loss, axis=-1)
```

2.4.5 Weighted Binary Cross Entropy:

Adding weights on standard.

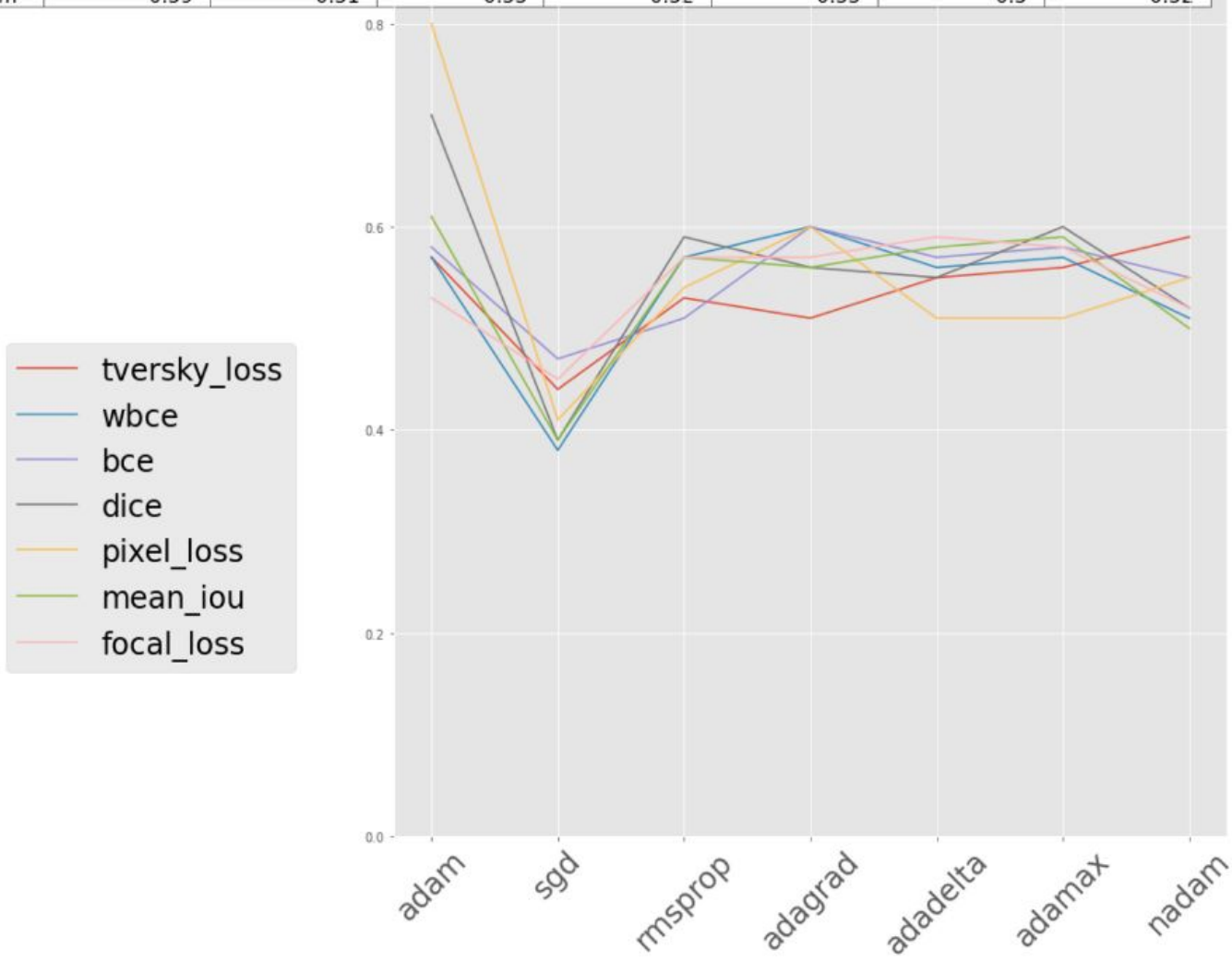
```
def weighted_bce_1(y_true, y_pred):
    weights = (y_true * 20.0) + 1.
    bce = tf.keras.backend.binary_crossentropy(y_true, y_pred)
    weighted_bce = tf.keras.backend.mean(bce * weights)
    return weighted_bce

def weighted_bce_2(y_true, y_pred):
    weights = (y_pred * 20.0) + 1.
    bce = tf.keras.backend.binary_crossentropy(y_true, y_pred)
    weighted_bce = tf.keras.backend.mean(bce * weights)
    return weighted_bce
```

2.4.6 Other loss functions:

Standard versions of all other loss functions were used.

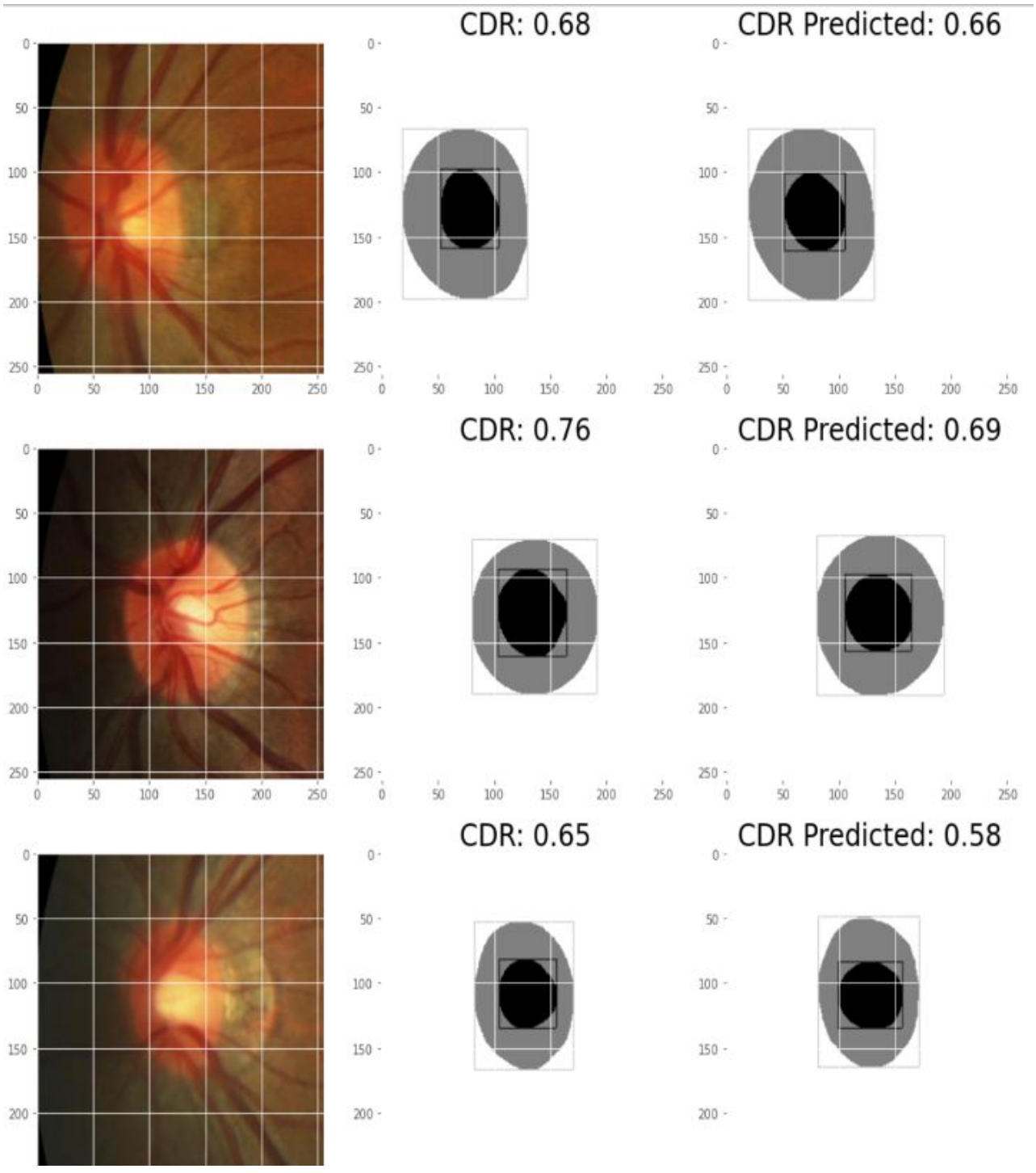
A comparison of optimizers and loss functions on one model is given in the figure below:

	tversky_loss	wbce	bce	dice	pixel_loss	mean_iou	focal_loss
adam	0.57	0.57	0.58	0.71	0.8	0.61	0.53
sgd	0.44	0.38	0.47	0.39	0.41	0.39	0.45
rmsprop	0.53	0.57	0.51	0.59	0.54	0.57	0.57
adagrad	0.51	0.6	0.6	0.56	0.6	0.56	0.57
adadelta	0.55	0.56	0.57	0.55	0.51	0.58	0.59
adamax	0.56	0.57	0.58	0.6	0.51	0.59	0.58
nadam	0.59	0.51	0.55	0.52	0.55	0.5	0.52



2.4.7 CDR Calculation:

In the end the metric that really mattered was the CDR. The model was trained to recognize discs and cups separately. Then in predict mode, the masks of both models are superimposed and CDR was calculated.



3. Model Performance Evaluation Criteria

3.1. Summary of Prediction Performance for Applied Models

Model version	Confidence Threshold	Cloud or Mobile	Images[1]	IoU / Jaccard	F1 / DICE	Avg Precision / Area under Precision Recall Curve
Civit & team's TPU based UNET (with dice loss)	N/A	Cloud	4600	0.65	0.81	N/A
Light U-Net v.1: 2 datasets with Dice loss function(Cup)	N/A	Cloud	209	0.72	0.78	N/A
Light U-Net v.1: 2 datasets with Dice loss function(Disc)	N/A	Cloud	209	0.88	0.76	N/A
Light U-Net v.2: 4 datasets with Dice + Binary Cross Entropy Loss function(Cup)	N/A	Cloud	694	0.65	0.78	N/A
Light U-Net v.3: 4 datasets with Dice Coefficient Loss function(Dice)	N/A	Cloud	704	0.88	0.76	N/A
Light U-Net v.3: 6 datasets with Dice + Binary Cross Entropy Loss function(Cup)	N/A	Cloud	1189	0.72	0.78	N/A
Segmentation U-net on GCP -- Custom Disc Segmentation Model	N/A	Cloud	400	0.85	0.88	0.89
AutoML Classification two labels	0.5	Cloud	5281	N/A	0.76	0.88
AutoML Classification two labels	0.5	Mobile	5281	N/A	0.71	0.80
AutoML Object Detection	0.5	Cloud	1070	N/A	N/A	0.96
AutoML Object Detection	0.5	Mobile	1070	N/A	N/A	0.94

[1] Includes data augmentation

3.2. Model Evaluation Metrics

There are several frequently-used performance criterion for assessing success in biomedical image segmentation: 1) the Intersection-over-Union score, the Dice score, and 3) pixel accuracy.

1. The Intersection-over-Union (IOU) score is defined as:

$$\text{Intersection-over-Union} = |A \cap B| \; / \; |A \cup B|$$

Jeremy Jordan, in a post titled *"Evaluating image segmentation models"* [9], writes, *"The **Intersection over Union (IoU)** metric, also referred to as the Jaccard index, is essentially a method to quantify the percent overlap between the target mask and our prediction output. This metric is closely related to the Dice coefficient which is often used as a loss function during training."*

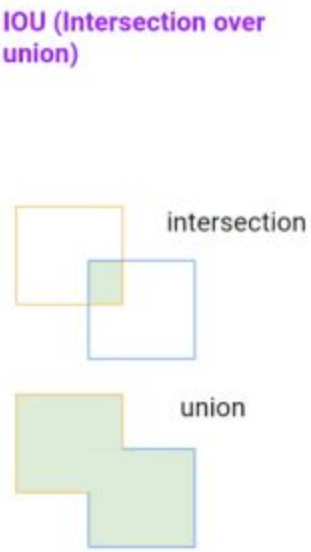
2. The Dice score or coefficient is defined as:

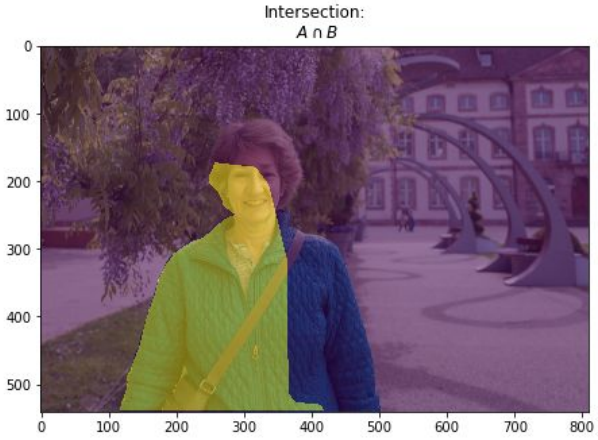
$$\text{Dice} = 2|A \cap B| \; / \; (|A| + |B|)$$

where "A" refers to our predicted mask, containing probabilities that each pixel belongs to the object of interest, and B refers to the "target" or ground truth mask and is a correct binary output map.

The Dice score is a similar but different measure of accuracy of the predicted segmentation to the ground truth segmentation. It is essentially the size of the overlap of the two segmentations divided by the total size of the two objects or masks.

The following figures illustrate the visualization of IOU components - intersection and union of the predicted versus the ground truth.





3. Pixel accuracy

Pixel accuracy is a third performance evaluation metric that classifies the pixels and measures the percentage of pixels correctly classified. The evaluation concept is similar to a binary classification where each pixel is classified as belonging to the mask, or not. Some of the performance metrics used in segmentation tasks as well as classification tasks are Precision & Recall, which are defined as follows:

- Precision = TP/(TP +FP)
- Recall = TP/(TP+FN)
- True Negative Rate = TN/(TN + FP)

TP = number of items correctly labeled as belonging to the positive class
TN = items with correct rejection

FP = items incorrectly labeled as belonging to the positive class.
FN = items incorrectly labeled as belonging to negative class.

The final accuracy is measured as:

$$\text{Accuracy} = TP + TN/(TP + TN + FP + FN)$$

4. Scalable TensorFlow Model Deployment on GCP

4.1. TensorFlow Serving as method of deployment

We summarize below the process of migrating and deploying a TensorFlow model to Google Cloud Platform, which is the cloud platform choice of our sponsor.

The process of model migration from a local environment or on a notebook environment (such as Colab notebook or Jupyter notebook) can be done with different approaches. We chose the TensorFlow Serving method as it is the more commonly advocated method for effectively scaling up model training and prediction of TensorFlow models in recent years.

TensorFlow Serving is an efficient, scalable, battle-tested model server for model deployment on GCP. As the number of your model versions grow and with your need to transition from one model version to another, or as your prediction service load grows over time, it is preferable to wrap your model in a service whose sole role is to make predictions, such that the rest of the infrastructure interacts with and queries this prediction service as needed and in effect decouples your model from the rest of your infrastructure.

4.2. Overview of Model Deployment Steps on GCP

Here is an outline of the main steps in transforming, saving and pushing a TensorFlow model to the GCP platform.

1) Convert our TensorFlow model into a SavedModel object to be used with TensorFlow Serving. SavedModel is the universal serialization format for TensorFlow models
2) Store and push our SavedModel to GCP's Storage bucket
3) Activate CloudAI's API for our account; create a new model version on Cloud AI

4) Create a new service account in Google IAM. This is an account that represents an application (not a user) that would ping the prediction service for a prediction.
5) To use the GCP prediction API we must first decode and serialize the test image.

```
def decode_img(img):
  ''' convert the compressed string to a 3D uint8 tensor '''
  img = tf.image.decode_jpeg(img, channels=3)
  img = tf.image.convert_image_dtype(img, tf.float32)
  img = tf.image.resize(img, [IMG_WIDTH, IMG_HEIGHT])
  return img

def get_test_data(test_fn):
    ''' decode and serialize the test image '''
    img2 = tf.io.read_file(test_fn)
    decoded_img = decode_img(img2)
    data = tf.keras.backend.eval(decoded_img)
    data = data.tolist()
    return data
```

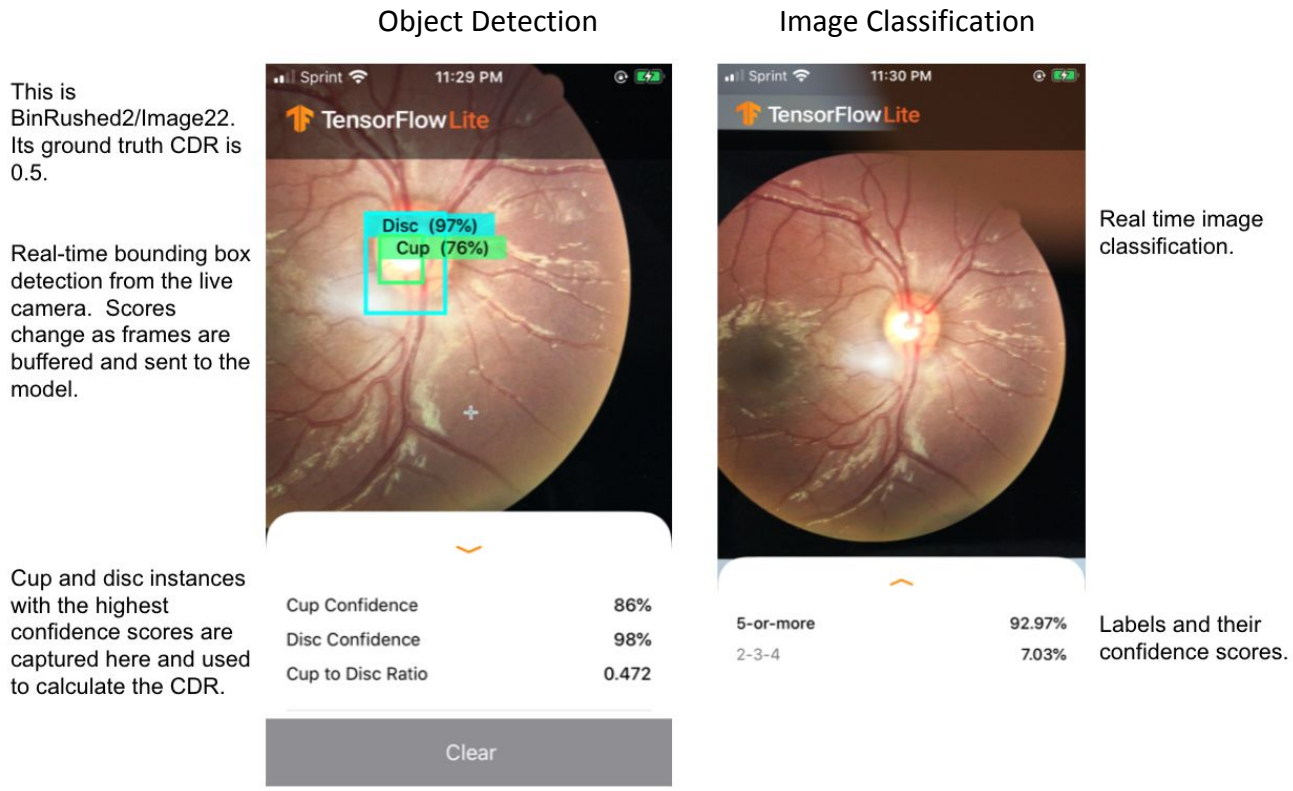
5. TensorFlow Lite Smartphone Deployment

TensorFlow Lite object detection and image classification models were exported from AutoML and embedded in an iOS application built with Swift. The application does not use a cloud-deployed model and therefore incurs no deployment costs. On-device deployment also delivers low-latency real time predictions.

The application uses the live camera feed, sending buffered frames to the model. The object detection model returns a series of instances of the cup and the disc along with localization details and confidence score. The cup and disc with the highest confidence score are selected for the CDR calculation which is persisted at the bottom.

Similarly the object detection model returns a series of instances of the predicted label and a confidence score.

Because the application uses the live camera feed, no findus images are ever captured or retained.



6. Ethical Considerations

- All of the images used in this project are publicly available and de-identified, and therefore are not protected by the HIPAA privacy rule[2].
- The models do not return any identifiable information as defined in the HIPAA privacy rule.
- The smartphone application does not store any images. It uses the live camera feed.
- A CDR measurement is not identifiable information as defined in the HIPAA privacy rule.
- These models and systems were created in the spirit of using "data for good".

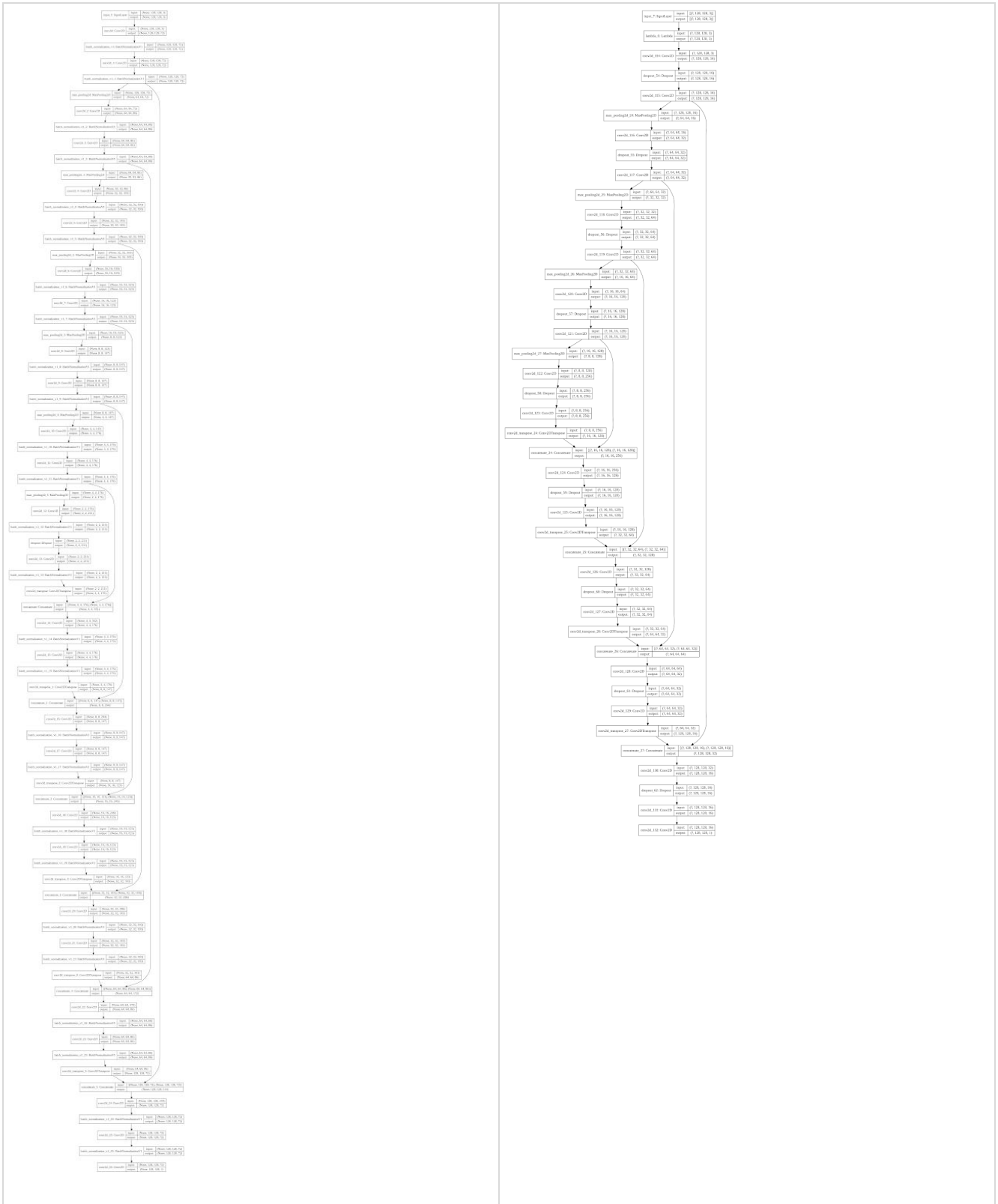
7. Summary

- Three main types of models were trained: U-Net, AutoML Object Detection and AutoML Image Classification. AutoML Object Detection was the best performing model with 94.83% recall and 99.61 precision%.
- Object detection was trained with over 1000 original fundus images.
- Image classification was trained with over 5000 images including augmentation.
- All three models are available on GCP for ad hoc cloud deployment and online predictions.
- A novel technique was developed for locating Disc Center for cropping images correctly for training.
- A novel technique was developed to chain loss functions during chaining to improve performance and accuracy.
- Generalized processing techniques were developed to generate masks and bounding boxes from all publicly available image datasets.
- The iOS application can run on iPhone or iPad. An ad hoc distribution has been provided to Johnson & Johnson for use on registered devices.
- Ground truth CDRs for all images have been provided to Johnson & Johnson.
- Data processing and model code is being turned over to Johnson & Johnson on GCP cloud storage.

[2] "HIPAA Privacy Rule and Its Impacts on Research." *National Institutes of Health*, U.S. Department of Health and Human Services, privacyruleandresearch.nih.gov/pr_08.asp.

Appendix

Graph 1: Model architecture graph for a deep U-Net by Civet et al. (left) vs. our lighter U-Net model



Code Repository

This is not a complete list but all code has been uploaded to the cds-2020-code-repository bucket.

Cloud Storage Bucket	Folder	Filename	Description
cds-2020-code-repository	python	prep_automl_dataset.py	Creates input file to AutoML
cds-2020-code-repository	python	augment_images.py	Generates the augmented images for use with AutoML
cds-2020-code-repository	python	predict_gcp_unet.py	Makes a prediction on the U-Net model if it is deployed.
cds-2020-code-repository	ios_app_distribution	ObjectDetection.ipa	Binary for ad hoc deployment (contact Janet to register a device)
cds-2020-code-repository	ios_app_distribution	image22.mp4	Video of iPhone app
cds-2020-code-repository	ios_app_distribution	automl_csv_export_data-image_CDR_wc4-2020-05-10T21_36_36.071Z_image_object_detection_1 2.csv	Dataset used to train object detection model deployed in iOS application.
cds-2020-code-repository	ios_app_distribution	ground_truth_median_cdr.csv	Ground truth CDR for RIGA and REFUGE datasets. For RIGA it is the median of the six.
cds-2020-code-repository	ios_app_distribution/ObjectDetection	Various	Code for the iOS object detection application. Includes TFLite model exported from AutoML.

AutoML Vision Datasets and Models

We trained many versions of the object detection and image classification models in AutoML. We list the most notable here.

Model Name	Description
image_CDR_wc4_20200409094052	Best performing object detection model
single_label_medi_20200426034358	Best performing image classification model

References

1. Sevastopolsky, Artem. (2017). Optic Disc and Cup Segmentation Methods for Glaucoma Detection with Modification of U-Net Convolutional Neural Network. Pattern Recognition and Image Analysis. 27. 10.1134/S1054661817030269.

2. Ronneberger, Olaf & Fischer, Philipp & Brox, Thomas. (2015). U-Net: Convolutional Networks for Biomedical Image Segmentation.

3. Munir, K., Elahi, H., Ayub, A., Frezza, F., & Rizzi, A. (2019). Cancer Diagnosis Using Deep Learning: A Bibliographic Review. Cancers, 11(9), 1235. https://doi.org/10.3390/cancers11091235

4. Nieradzik, Lars. Losses for Image Segmentation. https://lars76.github.io/neural-networks/object-detection/losses-for-segmentation/

5. On Dice-coefficient loss function vs cross-entropy. https://stats.stackexchange.com/questions/321460/dice-coefficient-loss-function-vs-cross-entropy

6. On Dice Loss + Cross Entropy. https://discuss.pytorch.org/t/dice-loss-cross-entropy/53194/9

7. Javier Civit and Anton Civit. (2019). TPU based UNET Segmentation. https://github.com/javicivit/TPU-UNET

8. W. Wang, K. Yu, J. Hugonot, P. Fua and M. Salzmann, "Recurrent U-Net for Resource-Constrained Segmentation," 2019 IEEE/CVF International Conference on Computer Vision (ICCV), Seoul, Korea (South), 2019, pp. 2142-2151.

9. Jordan, Jeremy. “Evaluating image segmentation models” 30 May. 2018, https://www.jeremyjordan.me/evaluating-image-segmentation-models

10. COCO Consortium. (n.d.). COCO Object Detection Challenge. Retrieved May 12, 2020, from http://cocodataset.org/#detection-eval

11. “HIPAA Frequently Asked Questions.” *American Psychological Association*, American Psychological Association, www.apa.org/science/programs/research/hipaa-faq.

12. “HIPAA Privacy Rule and Its Impacts on Research.” *National Institutes of Health*, U.S. Department of Health and Human Services, privacyruleandresearch.nih.gov/pr_08.asp.