I wanted to try using TensorFlow to make a model for this project so I watched about 7 different hour long videos on the topic of anomaly detection and used stack overflow way too many times.

1) To start, I needed to make the necessary imports:

```
#%%

import tensorflow as tf
import numpy as np
from tensorflow import keras
import os
#import cv2
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.preprocessing import image
import matplotlib.pyplot as plt
```

2) I used python to extract the abnormal and normal retinal images into separate folders so that the data was in a form that the ImageDataGenerator can utilize:

```
#%%
cwd = os.getcwd()

dataPath = cwd + '/Data'

trainDataLabels = pd.read_csv(dataPath + '/Training_Set/RFMiD_Training_Labels.csv')

trainDataLabels = trainDataLabels.iloc[:,:2]
trainDataLabels.tail()

Run Cell | Run Above | Debug Cell
#%%

#Adds column to df to have the image file name
trainDataLabels['Image_ID'] = trainDataLabels['ID'].astype(str) + '.png'

Run Cell | Run Above | Debug Cell
#%%
#Iterates through dataframe and splits the normal and abnormal retinal scans into two folders to train autoencoder only on the normal retinas

# iterate over the unique label
for label in trainDataLabels['Disease_Risk'].unique():
    if(label == 0):
        item_name = 'Normal'
    else:
        item_name = 'Abnormal'

    # create folder according to the label name
    item_folder = Path(dataPath + '/Training_Set/' + item_name + 'Training/')
    item_folder.mkdir(parents=True, exist_ok=True)

    # iterate over all possible number of unique labels
    for imageID in trainDataLabels.loc[trainDataLabels['Disease_Risk'] == label]['Image_ID']:

        img = Image.open(dataPath + '/Training_Set/Training/' + imageID) # read the image
        img.save(dataPath + '/Training_Set/' + item_name + 'Training/' + imageID) # and save to target folder
```

3) After doing this I created an ImageDataGenerator, with the rescale parameter essentially normalizing each pixel to be a value between 0 and 1 because normalization helps the model

train more effectively. The validation_split parameter just reserves 20% of the data to validate the competency of the model on unseen data that it hasn't been trained on. The train_generator acts as a gateway into the training folder to grab images in batches of 16 at a time, with a binary class_mode because I considered a binary classification model as a good approach for the project. The validation_generator will be used at the end of each epoch, or generation, during training to get the AUC and accuracy metrics on unseen data:

```
train = ImageDataGenerator(rescale=1/255, validation_split=0.2)

train_generator = train.flow_from_directory(
    dataPath + 'Training_Set/Images',
    target_size=(width, height),
    batch_size=16,
    class_mode='binary',
    subset = 'training'
    )

validation_generator = train.flow_from_directory(
    dataPath + 'Training_Set/Images',
    target_size=(width, height),
    batch_size=16,
    class_mode='binary',
    subset = 'validation'
    )
```

4) I understood that Convolutional Neural Networks (CNN) were the best approach for getting a model to understand images, so I researched how to actually architect one. I found a decently complex one to model mine off of and I started by creating a sequential model from keras. Then I began adding combinations of Conv2d and MaxPool2D layers. From my understanding the Conv2D layers act as a window view into the image that traverses along different sections of the image to gather features about each section. The MaxPooling2D extract the most prominent features in each section by selecting the max value in them. I used 4 of those layers arbitrarily, and then used a flatten layer along with a dense layer of 512 neurons. A dense layer simply means that each neuron in that layer connects to and influences every neuron in the next I'm not sure that I understood exactly what these layers were for, but I just saw a pattern of these being used in CNN's so I included it. The final layer is the output layer which has sigmoid activation which is just a way to squeeze the answer the model found into the range [0, 1], which is the binary classification answer to what the model predicts the image to be, abnormal or normal:

```
#%%

model = keras.Sequential()

# Convolutional layer and maxpool layer 1
model.add(keras.layers.Conv2D(32,(3,3),activation='relu',input_shape=(width,height,3)))
model.add(keras.layers.MaxPool2D(2,2))

# Convolutional layer and maxpool layer 2
model.add(keras.layers.Conv2D(64,(3,3),activation='relu'))
model.add(keras.layers.MaxPool2D(2,2))

# Convolutional layer and maxpool layer 3
model.add(keras.layers.Conv2D(128,(3,3),activation='relu'))
model.add(keras.layers.MaxPool2D(2,2))

# Convolutional layer and maxpool layer 4
model.add(keras.layers.Conv2D(128,(3,3),activation='relu'))
model.add(keras.layers.MaxPool2D(2,2))

# This layer flattens the resulting image array to 1D array
model.add(keras.layers.Flatten())

# Hidden layer with 512 neurons and Rectified Linear Unit activation function
model.add(keras.layers.Dense(512,activation='relu'))

# Output layer with single neuron which gives 0 for Abormal or 1 for Normal
#Here we use sigmoid activation function which makes our model output lie between 0 and 1
model.add(keras.layers.Dense(1,activation='sigmoid'))
```

5) The previous model ended up being far too complex for my laptop to process with this much data and parameters and would crash constantly, so I knew I had to simplify the architecture so that my dinky laptop can actually handle it. I reduced it to just 1 convolutional and max pooling layer to lighten the load. Which I know this isn't ideal and the model could have performed way better using a more complex architecture, but I tried so many times to get the previous model to work and my laptop wouldn't let it happen. So once I finished architecting this one, I compiled it. The optimizer is just the style of adjusting the weights between each neuron. The loss, binary_crossentropy, is a loss function that is used in binary classification models to score how well the model is doing and tell it when it is doing well. The metrics accuracy and AUC just show me how effective the model is during each epoch, or generation:

```
#%%
mod2 = keras.Sequential()

# Convolutional layer and maxpool layer 1
mod2.add(keras.layers.Conv2D(32,(3,3),activation='relu',input_shape=(width,height,3)))
mod2.add(keras.layers.MaxPool2D(2,2))

# This layer flattens the resulting image array to 1D array
mod2.add(keras.layers.Flatten())

mod2.add(keras.layers.Dense(1,activation='sigmoid'))

mod2.compile(optimizer='adam',loss='binary_crossentropy',metrics=['accuracy', 'AUC'])
```

6) Then I began to fit the model on the training data, and it took a few tries for it to properly work. I had to keep adjusting the steps per epoch so that it wouldn't run out of data to look at. From my understanding this parameter translates to how many batches of images the model will look at during each epoch. Each epoch is essentially a run through of the whole training dataset. It took HOURS for this simple architecture to be completed with only 3 epochs and was very frustrating:

```
#steps_per_epoch = train_imagesize/batch_size

mod2.fit(train_generator,
         steps_per_epoch = 96,
         epochs = 3,
         validation_data = validation_generator
         )
```

7) This shows the progress of the model throughout the training process getting slightly better through each epoch. The final AUC of the model on the training data was 0.8308 while on the validation set it was 0.7219. I'd say those are very nice metrics for such a simple model:

```
Epoch 1/3
96/96 [==============================] - 2404s 25s/step - loss: 1.8497 - accuracy:
0.7995 - auc: 0.7260 - val_loss: 0.5880 - val_accuracy: 0.7911 - val_auc: 0.7139
Epoch 2/3
96/96 [==============================] - 2321s 24s/step - loss: 0.4708 - accuracy:
0.8468 - auc: 0.7830 - val_loss: 0.4677 - val_accuracy: 0.7911 - val_auc: 0.7183
Epoch 3/3
96/96 [==============================] - 2315s 24s/step - loss: 0.3737 - accuracy:
0.8594 - auc: 0.8308 - val_loss: 0.4790 - val_accuracy: 0.7911 - val_auc: 0.7219

<keras.callbacks.History at 0x7f77b612f250>
```