Early Blindness & Diabetic Retinopathy Detection

A portable medical device that can accurately detect different stages of diabetic retinopathy, without the need for an internet connection, ample energy or computational power.
A portable medical device that could accurately detect different stages of diabetic retinopathy, without the need for an internet connection, would greatly reduce the number of cases of blindness due to retinopathy worldwide. With embedded machine learning, developing algorithms that can run directly on battery-powered medical devices and perform detection or diagnosis is now possible. In this article, we provide a walkthrough of the steps needed to quickly train an algorithm to deliver this capability using the software platform Edge Impulse.

Diabetic retinopathy is a condition in which damage occurs to the blood vessels in the tissues located at the back of the eye. It can occur in individuals who are diabetic and whose blood sugar is poorly managed. In extreme chronic cases, diabetic retinopathy can lead to blindness.

More than two in five Americans with diabetes have some form of diabetic retinopathy. That makes catching it early critical, at which point lifestyle or medical intervention can be performed. For rural areas around the world where access to vision care is limited, the stages of retinopathy are even harder to detect earlier before a case becomes severe.

Using diabetic retinopathy detection as the goal, we looked to take publicly available medical data and train a machine learning model in Edge Impulse that could run inference directly on an edge device. The algorithm would ideally be able to assess the severity of diabetic retinopathy between images of eyes taken by a retinal camera. The dataset that we used for this project can be found here.
Various images of eyes showing different stages of diabetic retinopathy.

For this algorithm, we divided the classes into five different datasets: No diabetic retinopathy (No DR), Mild DR, Moderate DR, Severe DR and Proliferative DR. As with many publicly available datasets, some data cleansing and labeling had to be done.

To protect patient identities, each image in the dataset was simply given an id_code and a diagnosis from 0-5, 0 being the lowest severity of No DR and 5 being the worst, or Proliferative DR.

<table>
<thead>
<tr>
<th>id_code</th>
<th>diagnosis</th>
</tr>
</thead>
<tbody>
<tr>
<td>000c1434d8d7</td>
<td>2</td>
</tr>
<tr>
<td>001639a390f0</td>
<td>4</td>
</tr>
<tr>
<td>0024cdab0c1e</td>
<td>1</td>
</tr>
<tr>
<td>002c21358ce6</td>
<td>0</td>
</tr>
<tr>
<td>005b95c28852</td>
<td>0</td>
</tr>
<tr>
<td>0083ee8054ee</td>
<td>4</td>
</tr>
<tr>
<td>0097f532ac9f</td>
<td>0</td>
</tr>
<tr>
<td>00a8624548a9</td>
<td>2</td>
</tr>
<tr>
<td>00b74780d31d</td>
<td>2</td>
</tr>
</tbody>
</table>

Organization of the open source diabetic retinopathy dataset, which allows patients’ data privacy to be preserved.

In order to ingest the data into Edge Impulse, some partitioning of the images needed to occur. Given the simple nature of how the data was divided, I decided to write a VBA script to read the image id_code from Excel, grab the associated image, and put it in its respective folder. The script to move these files is linked here. For people more savvy with Python or other scripting languages, there are many ways to do this, which might even be simpler!
Folder organization of datasets to make ingestion into Edge Impulse easy.

Edge Impulse has other data ingestion features such as cloud data bucket integration or data collection from devices, but data upload was the method I utilized here. Using the data upload option, I was able to bring in my 5 different classes a series of five uploads. Each upload consisted of me labeling the data as one of the 5 classes, and uploading the associated images contained in each folder.
Edge Impulse has the option to automatically split data into training or testing data with an 80/20 split. However, I manually added about 500 images across the different classes to the test dataset.

Next, it was time to set up my model and choose the signal processing block + neural network block for this model. For this model, I fed the image block into a transfer learning block with the goal of differentiating between five different classes.

![Image of Edge Impulse interface]

The signal processing and machine learning pipeline visually shown in Edge Impulse.

From here, I went to train the neural network. Playing around with the settings of the neural network, the best accuracy I was getting was around 74%. Not bad, but the model was getting stuck when it came to some of the edge cases. For example, severe DR was sometimes getting classified as mild DR. The model was not super accurate as DR progressed, as you can see in the screenshot below.
Result from the trained model in Edge Impulse, for detecting different stages of diabetic retinopathy (DR). This iteration yielded a 74.1% accurate model for detecting the 5 different stages of DR, with a loss of 0.83.

This made me think about the real-life applications of a project like this and if this level of accuracy would be acceptable. Ideally, some sort of portable retinal imaging camera (in a low wireless connectivity environment) could run an algorithm like this on the device itself. When the picture is taken, processed and a result output, at that point the person administering the eye test could tell the patient that they need to go seek further medical help or intervention, depending on the result.

For this application, it is more important to catch DR at all stages so the patient can either begin some preventative treatment, or for more severe cases, seek immediate medical help. Given this use case, the model actually serves its potential application relatively well.

Off the top of my head, there are a few changes or improvements that I could make to the model, that might make the resulting output more accurate in terms of diagnosing the severity of DR:
• More data is always better! However, given this limited dataset, further data collection would be needed.
• One idea could be to combine classes, by creating a mild - moderate class and a proliferative - severe class. I wonder if that might help the algorithm better classify, given the similarities between certain cases of mild and moderate DR, which would now all fall into the same group.
• Play around with the number of layers within the neural network (NN), as well as the dropout.

The neural network architecture section, in which layers of the neural network can be edited.

From a deployment perspective, this trained model did have a larger footprint in terms of memory, taking up an estimated 306kB of Flash and 236kB of RAM. Depending on the device selected to run inference on, the amount of time needed for an inference result to be provided back was anywhere from 0.8 seconds to 6 seconds, when benchmarking on either a Cortex-M4 at 80MHz or Cortex-M7 at 216MHz. Given this end product would need to be taking images however, I anticipate something like the Cortex-M7’s processing capabilities or higher would be needed.

In summary, using an open source dataset, we were able to train a relatively well-functioning machine learning model for detecting various forms of diabetic retinopathy (DR). The end goal would be to deploy models like this directly on the embedded microcontroller or Linux device, and have more medical devices like the one below run inference on the edge. This opens up new possibilities for healthcare services, by providing medical technology that can be used in rural areas, with no wireless connectivity to provide testing to populations that have low access to healthcare.