Automated diagnosis of Retinopathy of Prematurity Using Neural Networks

Shashank Rammoorthy
Sachin Rammoorthy

Objective

Retinopathy of Prematurity, or ROP, is a curable disease that leads to blindness in infants if not corrected within an exceedingly short window. Diagnosis must be done within a couple of weeks of birth, but this is often not possible in rural India due to both a lack of equipment and a lack of specialists. This project is an attempt at implementing an automated, deep learning based diagnosis of ROP.

Synopsis

ROP occurs in over 16% of all premature births. In babies weighing less than 1,700 grams at birth, over 50% will develop ROP. It is a condition that can lead to the development of impaired vision and blindness. It is a potentially blinding disease caused by abnormal development of retinal blood vessels in premature infants.

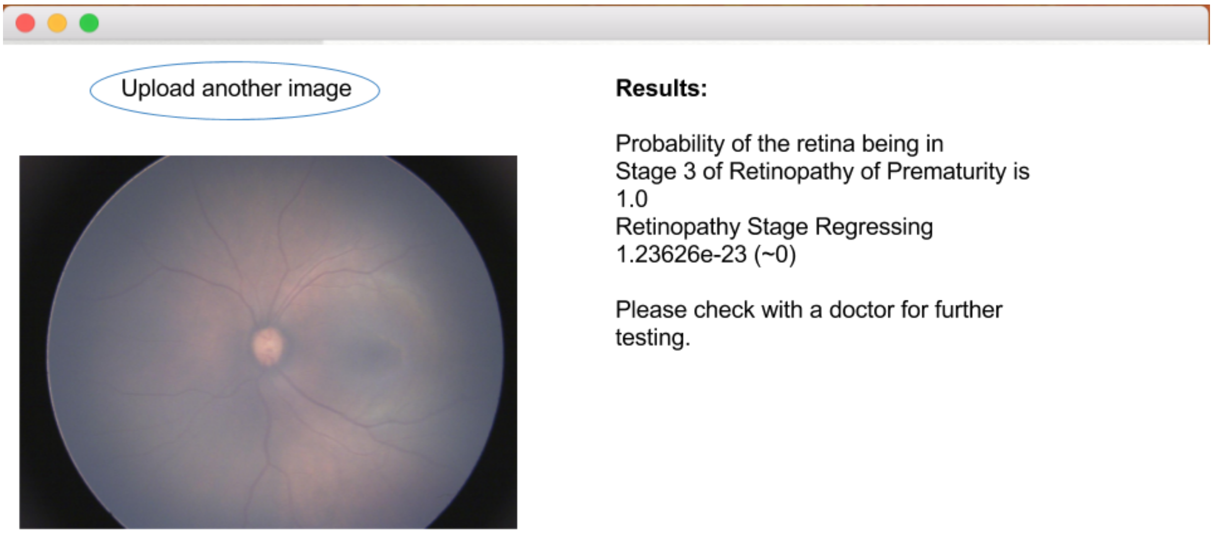
In this project, a neural network was retrained using transfer learning. The model created was trained on the 2012 ImageNet Large Visual Recognition Challenge datasets. The retrain script that I created retrained MobileNet - and not Inception V3, owing to time constraints on my local machine.

Nevertheless, on the test dataset, high levels of accuracy were obtained - 100% using MobileNet. This, however, was with a relatively small dataset. The training and test datasets I will be using will be much bigger in the next few months with access to more data. The data I used was closed and not open source.

I got an accuracy of >98% on the test set - but this will be improved with successive trials on more images over time.

Materials and Methods

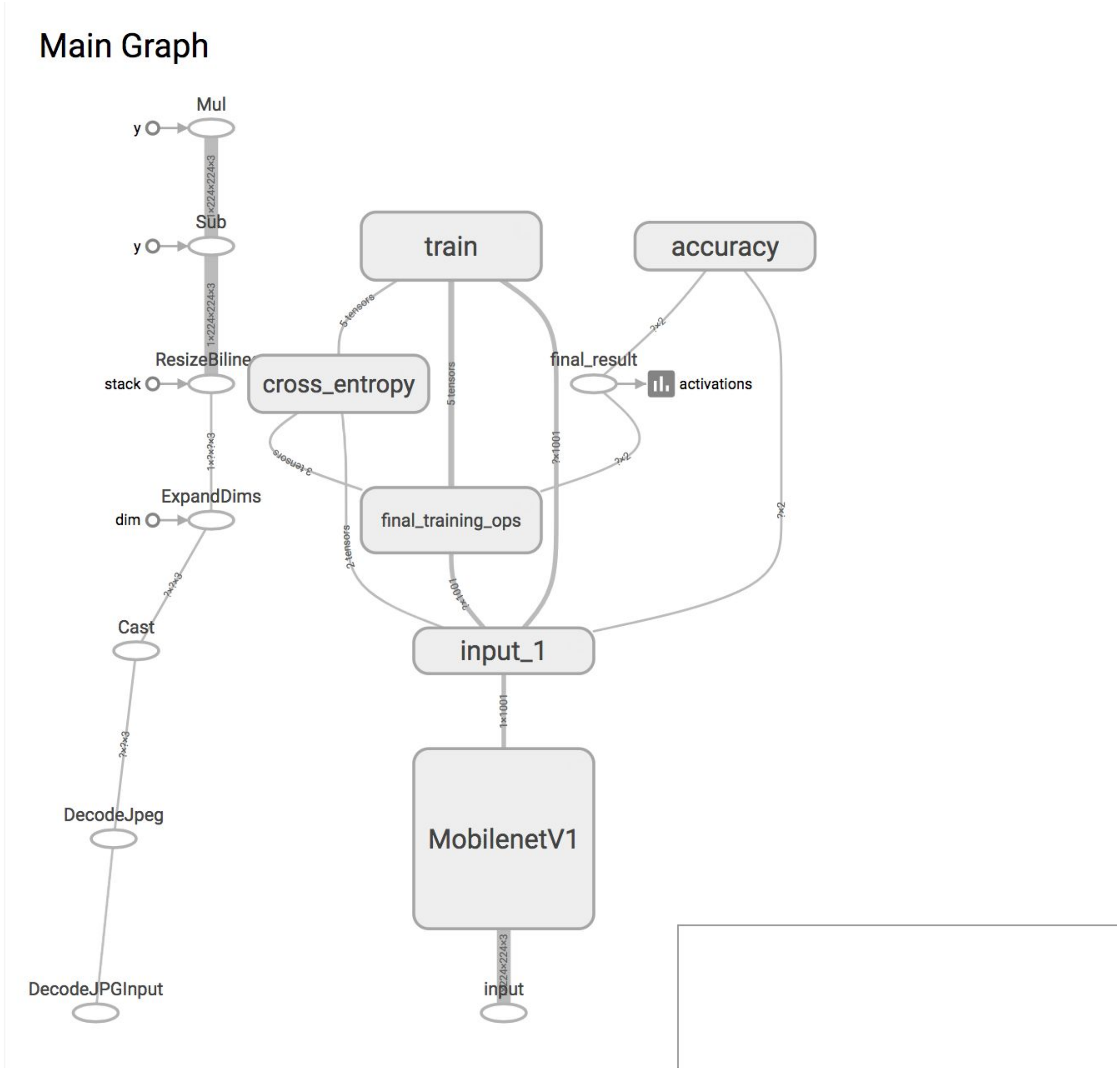
The desktop application was made using Electron, an open source framework for creating native applications using JavaScript. Below is a screenshot of the desktop application:



Desktop application

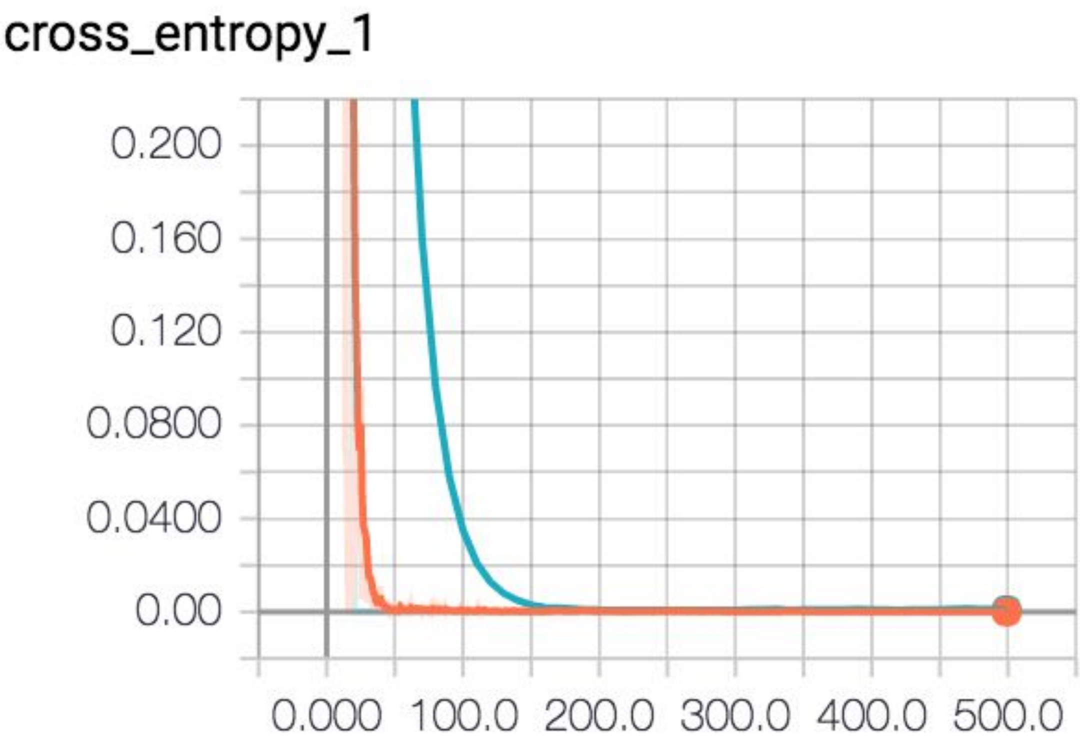
I used the Google MobileNet Neural Network and retrained it to identify differences in the available images in the dataset I was provided with, divided in a 80:10:10 train/test/validation ratio.

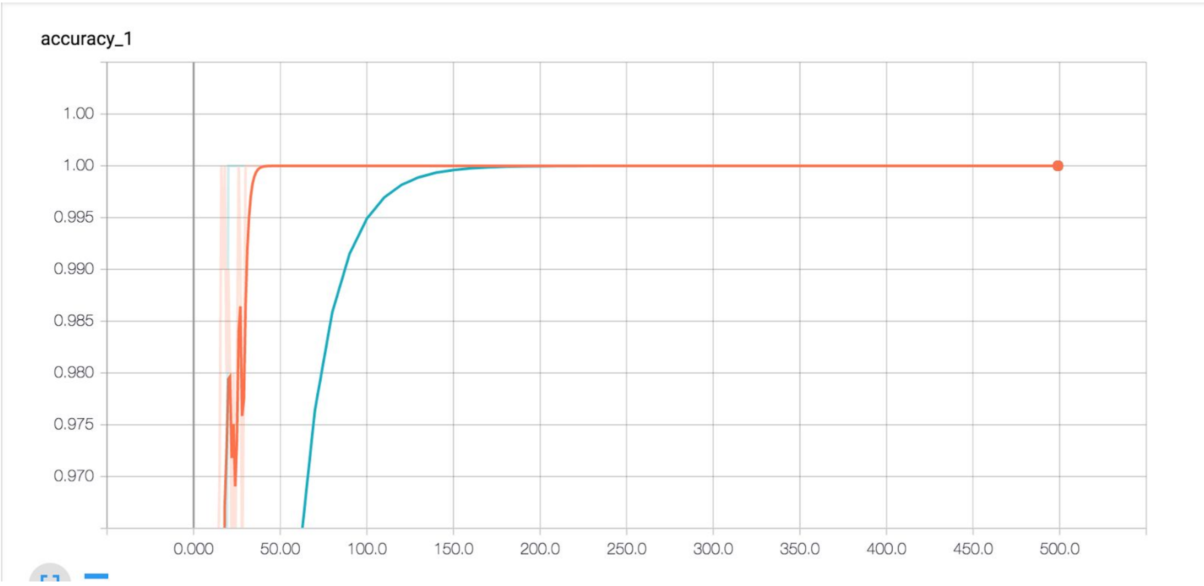
The input image resolution was 224 px, and the architecture that was used was MobileNet 0.50. There were 4000 training steps run in the script.



All this was done using Tensorflow, an open source library for training Neural Networks.

Cross-entropy over time is shown below:





Code extracts

label_image.py

```
from __future__ import absolute_import
from __future__ import division
from __future__ import print_function

import argparse
import sys

import numpy as np
import tensorflow as tf

def load_graph(model_file):
  graph = tf.Graph()
  graph_def = tf.GraphDef()

  with open(model_file, "rb") as f:
    graph_def.ParseFromString(f.read())
  with graph.as_default():
    tf.import_graph_def(graph_def)

  return graph

def read_tensor_from_image_file(file_name, input_height=299, input_width=299,
                                input_mean=0, input_std=255):
  input_name = "file_reader"
  output_name = "normalized"
  file_reader = tf.read_file(file_name, input_name)
  if file_name.endswith(".png"):
    image_reader = tf.image.decode_png(file_reader, channels = 3,
                                       name='png_reader')
  elif file_name.endswith(".gif"):
    image_reader = tf.squeeze(tf.image.decode_gif(file_reader,
                                                  name='gif_reader'))
  elif file_name.endswith(".bmp"):
    image_reader = tf.image.decode_bmp(file_reader, name='bmp_reader')
  else:
    image_reader = tf.image.decode_jpeg(file_reader, channels = 3,
                                        name='jpeg_reader')
  float_caster = tf.cast(image_reader, tf.float32)
  dims_expander = tf.expand_dims(float_caster, 0);
  resized = tf.image.resize_bilinear(dims_expander, [input_height, input_width])
  normalized = tf.divide(tf.subtract(resized, [input_mean]), [input_std])
  sess = tf.Session()
  result = sess.run(normalized)

  return result

def load_labels(label_file):
  label = []
  proto_as_ascii_lines = tf.gfile.GFile(label_file).readlines()
  for l in proto_as_ascii_lines:
```

```python
    label.append(l.rstrip())
  return label

if __name__ == "__main__":
  file_name = "tf_files/flower_photos/daisy/3475870145_685a19116d.jpg"
  model_file = "tf_files/retrained_graph.pb"
  label_file = "tf_files/retrained_labels.txt"
  input_height = 224
  input_width = 224
  input_mean = 128
  input_std = 128
  input_layer = "input"
  output_layer = "final_result"

  parser = argparse.ArgumentParser()
  parser.add_argument("--image", help="image to be processed")
  parser.add_argument("--graph", help="graph/model to be executed")
  parser.add_argument("--labels", help="name of file containing labels")
  parser.add_argument("--input_height", type=int, help="input height")
  parser.add_argument("--input_width", type=int, help="input width")
  parser.add_argument("--input_mean", type=int, help="input mean")
  parser.add_argument("--input_std", type=int, help="input std")
  parser.add_argument("--input_layer", help="name of input layer")
  parser.add_argument("--output_layer", help="name of output layer")
  args = parser.parse_args()

  if args.graph:
    model_file = args.graph
  if args.image:
    file_name = args.image
  if args.labels:
    label_file = args.labels
  if args.input_height:
    input_height = args.input_height
  if args.input_width:
    input_width = args.input_width
  if args.input_mean:
    input_mean = args.input_mean
  if args.input_std:
    input_std = args.input_std
  if args.input_layer:
    input_layer = args.input_layer
  if args.output_layer:
    output_layer = args.output_layer

  graph = load_graph(model_file)
  t = read_tensor_from_image_file(file_name,
                                  input_height=input_height,
                                  input_width=input_width,
                                  input_mean=input_mean,
                                  input_std=input_std)

  input_name = "import/" + input_layer
  output_name = "import/" + output_layer
  input_operation = graph.get_operation_by_name(input_name);
  output_operation = graph.get_operation_by_name(output_name);

  with tf.Session(graph=graph) as sess:
    results = sess.run(output_operation.outputs[0],
                      {input_operation.outputs[0]: t})
```

```
results = np.squeeze(results)

top_k = results.argsort()[-5:][::-1]
labels = load_labels(label_file)
for i in top_k:
  print(labels[i], results[i])
```

Results/Observations/Findings

The KIDROP organisation says that for every 100 infants they screen, about 20-40 will have ROP in some stage, and 2-4 will need treatment to avoid blindness.

I cannot distinguish between pictures of an infected eye in Stage 3 and an unaffected eye. The differences are often subtle - and it almost requires the presence of an ophthalmologist, either physically or through telemedical methods.

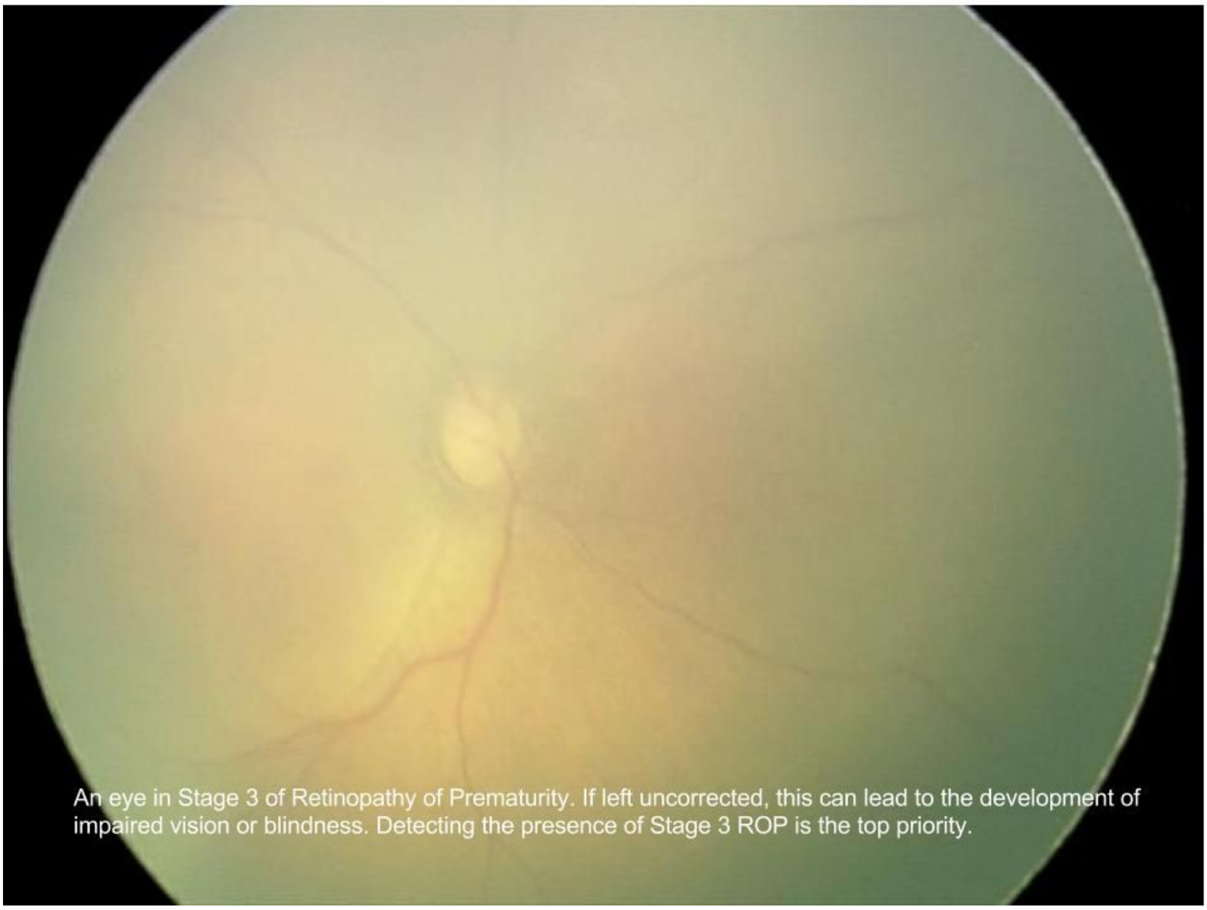
In the future with the access of a public API for the images, I will be creating a neural network with even greater accuracy for the same task.

Innovation

There have been machine learning approaches in the past to help detect ROP in infants, however, these have all used machine learning approaches, as opposed to deep learning and Convolutional Neural Networks as used in this project.

Among others, Google Research has come up with a deep learning neural network to aid in the diagnosis of diabetic retinopathy - but nobody has implemented the same for Retinopathy of Prematurity, arguably a more urgent and pressing condition.

Images have been uploaded, but here they are:



An eye in Stage 3 of Retinopathy of Prematurity. If left uncorrected, this can lead to the development of impaired vision or blindness. Detecting the presence of Stage 3 ROP is the top priority.

Acknowledgements

I would like to thank Dr Anand Vinekar and KIDROP for providing me with the sample images for me to use as the dataset. I hope this project is of use to them and countless others.

I would also like to thank Dr Padma Rammoorthy, for her help from a medical standpoint, and for helping with ideation and planning.

References

Abadi, Martín, et al. "Tensorflow: Large-scale machine learning on heterogeneous distributed systems." *arXiv preprint arXiv:1603.04467* (2016).

Chiang, Michael F. et al. "Image Analysis for Retinopathy of Prematurity Diagnosis." Journal of AAPOS : the official publication of the American Association for Pediatric Ophthalmology and Strabismus / American Association for Pediatric Ophthalmology and Strabismus 13.5 (2009): 438–445. PMC. Web. 5 Sept. 2017.

John D. Hunter. **Matplotlib: A 2D Graphics Environment**, Computing in Science & Engineering, **9**, 90-95 (2007),DOI:10.1109/MCSE.2007.55

Jones E, Oliphant E, Peterson P, *et al.* **SciPy: Open Source Scientific Tools for Python**, 2001-, http://www.scipy.org/ [Online; accessed 2017-08-15].

"Python Anywhere." *Python Anywhere*, Python Anywhere, PythonAnywhere.com.

Rosebrock, Adrian. "Creating a Face Detection API with Python and OpenCV ."*PyImageSearch*, 7 June 2015.

Stéfan van der Walt, S. Chris Colbert and Gaël Varoquaux. **The NumPy Array: A Structure for Efficient Numerical Computation**, Computing in Science & Engineering, **13**, 22-30 (2011), DOI:10.1109/MCSE.2011.37

Stéfan van der Walt, Johannes L. Schönberger, Juan Nunez-Iglesias, François Boulogne, Joshua D. Warner, Neil Yager, Emmanuelle Gouillart, Tony Yu and the scikit-image contributors. **scikit-image: Image processing in Python**, PeerJ 2:e453 (2014)