

# Diabetic Retinopathy Prediction using Convolutional Neural Network (CNN) using low-power devices.

[Satish Swarnkar](#) ✉

---

Artificial Intelligence: Deep Learning, Human-Centered AI, and Beyond, Stanford University

## 1. Abstract:

Diabetic Retinopathy (DR) is a leading cause of vision loss around the world. The goal of this project is to apply deep learning (DL) concepts of Artificial Intelligence to build a real-life application that can run a low-power device, while experimenting with various combinations of datasets, convolutional neural network (CNN) models, hyperparameters and predict the severity of diabetic retinopathy.

We obtained thousands of labeled Color Fundus Retinal Photographs (CFPs) from Kaggle<sup>(2)</sup> for training and classification. Initially, the predictions were about 20% accurate. To improve results we balanced the dataset, pre-processed, trained all layers of the model, and searched for Hyper-Parameters using HyperOpt<sup>(6)</sup>.

Finally, using Android Studio we built an app to run on an Android phone to continuously scan and predict the Diabetic Retinopathy level.

We were able to successfully build an application to predict diabetic retinopathy. We are confident that additional research, training with



*Image of a portable retinal imaging system. Source <sup>(8)</sup>*

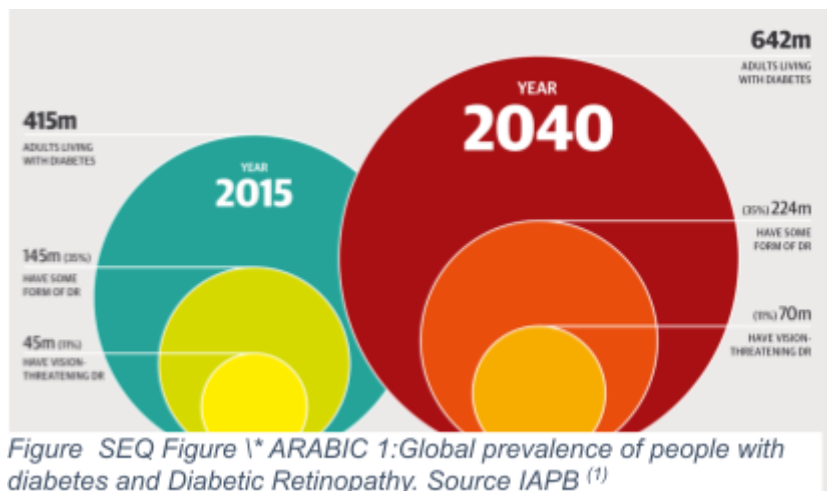
larger high-quality data, and improved models can yield even more accurate results and potentially be integrated with the camera and deployed to prevent blindness.

## 2. Table of Contents

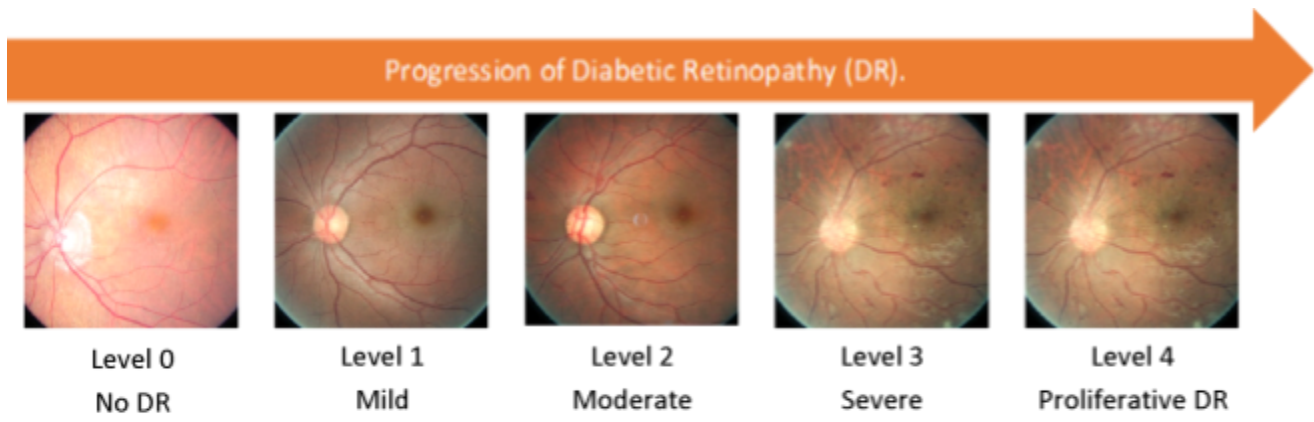
1. Abstract:	1
2. Table of Contents	2
3. Introduction	2
4. Project Structure	4
4.1. Step-by-step Machine Learning for Image Classification:	4
4.2. Datasets	5
5. Result	6
5.1. Training with Diabetic Retinopathy	6
5.2. Training with Tiny ImageNet dataset with Animal images	8
6. Analysis of the Results	9
7. Conclusion	10
8. References	10
9. Acknowledgements	10

## 3. Introduction

As per IAPB (International Agency for the Prevention of Blindness), in 2015, out of 415 million people living with diabetes over 145 million had diabetic retinopathy (DR) and that number is expected to grow to 224 million by the year 2040<sup>(1)</sup>. Early identification and treatment can prevent almost all blindness; however, DR often remains undetected until it gets too severe. Ophthalmologists examine CFPs to document the presence, progression, and severity



of disorders. Unfortunately, a large population does not have access to these experts and hence many patients remain undiagnosed and lose their vision.



In this project, we evaluated if deep learning, specifically convolutional neural networks (CNN) can be used for assessment and prediction of diabetic retinopathy. The purpose of this project was to focus on three key areas to get optimal results – Data sets, deep learning models, and AI inference.

- 1) Datasets: We used Retinal Photographs and 5 classes of animal images as inputs to the models.
- 2) Models: We tried numerous models with combinations of feature learning layers (Convolutions and Pooling) and classification layers (flattened, fully connected layers). In addition, we used VGG16<sup>(4)</sup> and ResNet<sup>(5)</sup> models to obtain baseline results. Finally, we studied the influence of hyperparameters on overfitting and underfitting training curves as well as stride lengths and epochs. We used Google Colab, Jupyter, Tensorflow, and Python for the development.
- 3) AI inference on battery-powered devices: We exported trained models and used the TensorFlow Lite interpreter. To perform inference we used an Android phone camera to capture an image of the retina and passed that to the model as an input. The model then showed an array of probabilities

between 0 and 1 for each class or level of the DR. We used TensorFlow Lite<sup>(7)</sup> Java API to perform the Inference.

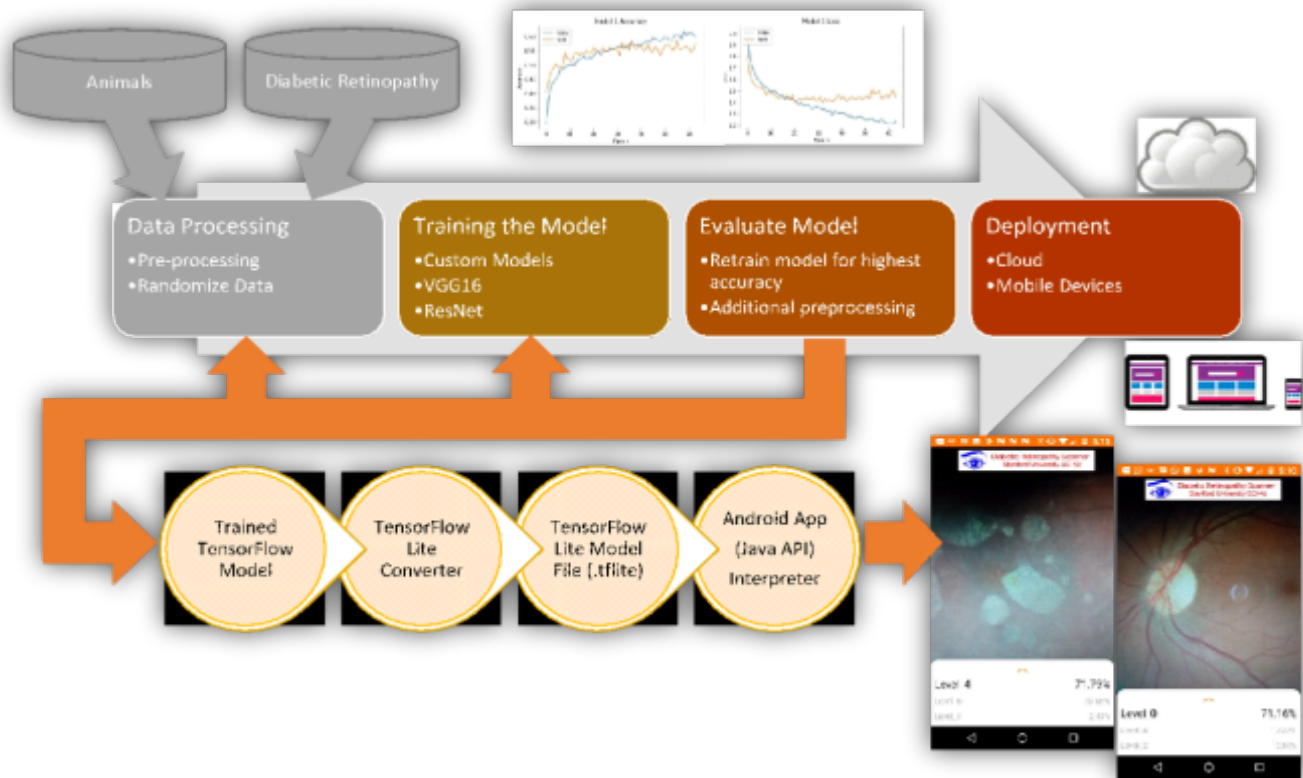


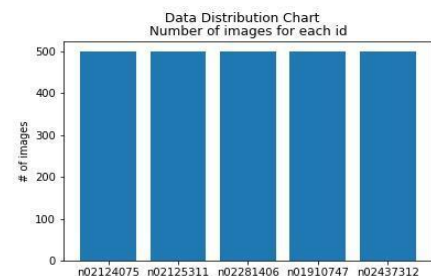
Figure 3: Project Flow: stages of the project from loading data sets to building an AI application to run on Android phone

## 4. Project Structure

### 4.1. Step-by-step Machine Learning for Image Classification:

#### 4.1.1. Data Processing and Training the Models

- 1) Load the data from local storage or Google Drive. Plot a histogram with the class and number of images for each class

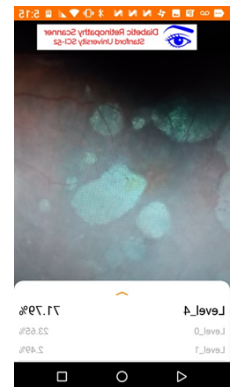


- 2) Preprocess the data: Resize to 224x224 pixels, convert to RGB, shuffle all data, and split into training and validation sets.
- 3) Create various CNN Models (with ReLU activation function and different stride lengths).
- 4) Train the model with different epochs and batch sizes. Plot accuracy and losses for each run
- 5) Evaluate the model with a test data set and print the loss and accuracy results.

#### 4.1.2. AI Inferencing on Android Phone using TensorFlow Lite<sup>(7)</sup>

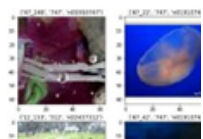
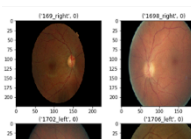
As a first step, export the trained model into TensorFlow Lite format, and the labels into a text file.

1. Get camera input
2. Classify
  - a) Load model, labels and instantiate Tensorflow Lite interpreter
  - b) Convert camera bitmap image to a TensorImage format and pre-process it
  - c) Allocate Tensor Buffer for output of the model
  - d) Run TensorFlow Lite inference, and get a list of recognitions (labels and confidence)
3. Display Results, with the highest confidence recognition on the top.



## 4.2. Datasets

We used two different datasets. Sample images are shown below. Each dataset was equally divided into various classes. For example, retina images of 224x224 pixels were classified as Level 0 (No DR) through Level 4 (Proliferative). The animal images (part of the Tiny ImageNet dataset) had 5 classes - cats, dogs, lions, camels, and panthers and each image was 64x64 pixels



## 5. Result

### 5.1. Training with Diabetic Retinopathy

The following tables summarize the results of the 4 models trained and tested on the unbalanced diabetic retinopathy dataset. The detailed runs and the outputs of all models are presented in Appendix 1.

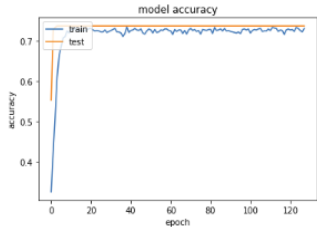
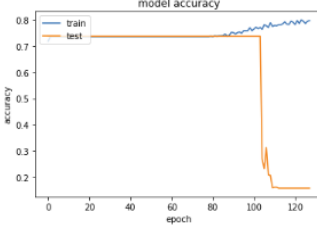
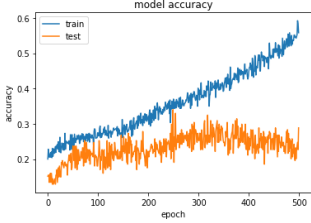
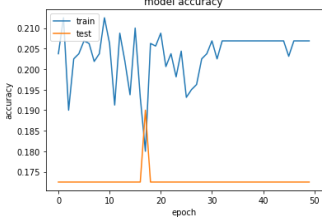
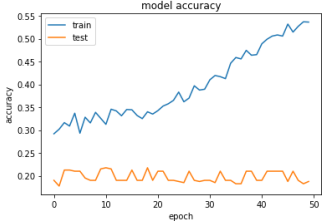
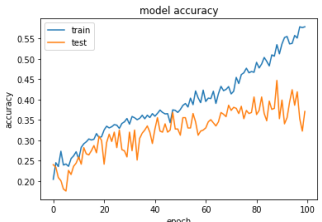
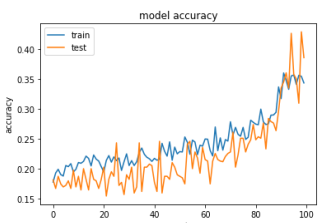
No	Architecture	Images	Training (Test Accuracy Plot)	Accuracy	Comments
1, 2, 3	Simple (4 Conv, 2 MP, 1DO, 2 FCN & Stride 2 and 3)  VGG16 (1 FCN + 1 DO)	1600 DR UnBalanced (1200 in Class 0)	128 epochs, Adam(lr=1e-05) 	74% Accuracy	75% of the images are class 0 (high bias expected) The classifier is almost always biased
4	ResNet 50 (Seq 4) 3 FCN	1600 DR UnBalanced (1200 Class 0)	128 epochs, Adam(lr=1e-05) 	16% Accuracy	75% of the images of class 0 High training and test loss divergence at epoch 100

Table 1: Describes the experiments run with an unbalanced dataset and training of the last few layers

Now, we look at tests with variations in training various layers with dataset balancing & pre-processing.

The detailed results are in Appendix 2.

No	Architecture	Images	Training (Test Accuracy Plot)	Accuracy	Comments
1	<a href="#">Simple Convolution</a> 6 Conv, 1 MPI & 2 FCN	1000 DR Balanced 200 images per class	500 Epochs, SGD(lr=0.0001)	25% Average Precision	Overfitting after 300 epochs Learning stopped at 25% precision

					
2	<a href="#">VGG-16 Imagenet weights</a> Trainable 4 layers & 2 FCN	2000 DR Balanced  400 Images per class	50 Epochs, Adam(lr=0.001) 	03% Average Precision	DR Features are different from imagenet.  No features are learned
3	<a href="#">Resnet50 Imagenet weights</a> Trainable 6 layers & 4 FCN	2000 DR Balanced  400 Images per class	50 Epochs, Adam(lr=0.001) 	09% Average Precision	Overfitting on train data  No features learned that could classify the test data
4	<a href="#">VGG-16 Imagenet Init</a> Train all, 7 FCN & 1 DO	2000 DR Bal/Crop  400 Images per class	100 Epochs, SGD(lr=0.0001) 	33% Average Precision	Overfitting on training after 60 epochs.  Accuracy on test data plateaued after 60
5	<a href="#">ResNet50 Imagenet Init</a> Train all, 7 FCN & 1 DO	2000 DR Bal/Crop  400 Images per class	100 Epochs, SGD(lr=0.0001) 	30% Average Precision	Train longer for convergence. Accuracy on test data follows training. Early stopping at epoch 100
6	<a href="#">Hyper-Parameter Search</a> Using Hyper-Opt on VGG16 training all layers	2000 DR Bal/Crop  400 Images	10 Epochs, 100 Max Trials HP Search Space LR = [ 0.0001, 0.001 ] Unit1= [ 1024, 2048 ] Unit2= [ 1024, 2048 ]	33% Precision HP Search  BatchSize = 8 LR = 0.0001	Hyper-Opt searched the space provided and the proposed results

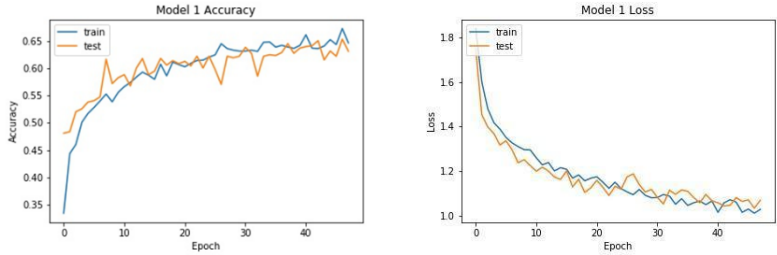
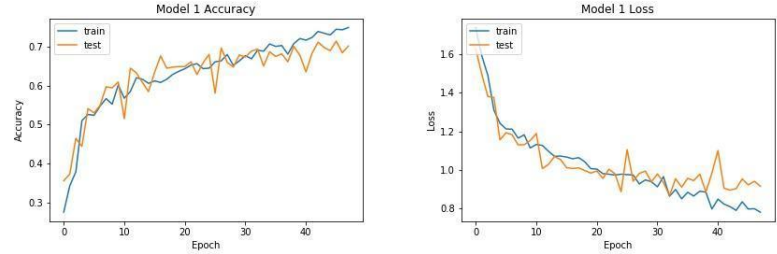


	with image crop	per class	DO1 = (.45,.50) Optimizer=['Adam','SGD','RMS prop','Adadelata'] BatchSize =[6, 8] Activation =['relu']	DO1 = 0.5 Unit1 = 1024 Unit2 = 1024 Optimizer=SGD	were used in Arch 4 training for sub-100 epoch comparison
--	-----------------	-----------	---	--	---

Table 2: Describes the experiments run with a balanced dataset and training of all/selected layers

## 5.2. Training with Tiny ImageNet dataset with Animal images

As the accuracy with diabetic retinopathy images was not very high, we decided to experiment with the Tiny ImageNet dataset and picked up 5 animal classes for apples-to-apples comparison. The following table summarizes the results of the 3 of the many models trained and tested on the Tiny ImageNet data. The detailed runs and the outputs of all models are presented in Appendix 3.

No	Architecture	Images	Training Results (Epochs, gradient descent optimization algorithms, Accuracy and Loss), and comments
1	Convolutional Model (2 Conv, 1 Pooling, 2 Conv, 1 Pooling, Flatten, 6 Dense, 1 Dropout, 1 Dense (5 classes) Activation: ReLU	2500 (1750 for Training and 750 for test) Balanced across classes	48 Epochs, Adam(lr=1e-05), 63% Accuracy  <p>The figure contains two line plots. The left plot, titled 'Model 1 Accuracy', shows training (blue) and test (orange) accuracy over 48 epochs. The training accuracy starts at approximately 0.35 and rises to about 0.63, while the test accuracy starts at 0.45 and rises to about 0.60. The right plot, titled 'Model 1 Loss', shows training (blue) and test (orange) loss over 48 epochs. Both training and test loss start at approximately 1.7 and decrease to around 1.05 by epoch 48.</p>
2	Convolutional Model (2 Conv, 1 Pooling, 2 Flatten, 12 Dense, 1 Dropout, 1 Dense (5 classes) Activation: ReLU	2500 (1750 for Training and 750 for test) Balanced across classes	48 Epochs, Adam(lr=1e-05), 70% Accuracy  <p>The figure contains two line plots. The left plot, titled 'Model 1 Accuracy', shows training (blue) and test (orange) accuracy over 48 epochs. The training accuracy starts at approximately 0.3 and rises to about 0.70, while the test accuracy starts at 0.4 and rises to about 0.68. The right plot, titled 'Model 1 Loss', shows training (blue) and test (orange) loss over 48 epochs. Both training and test loss start at approximately 1.6 and decrease to around 0.9 by epoch 48.</p>
3	Convolutional Model (2 Conv, 1 Pooling, 2 Conv, 1	2500 (1750 for Training	64 Epochs, Adam(lr=1e-05), 74.6% Accuracy

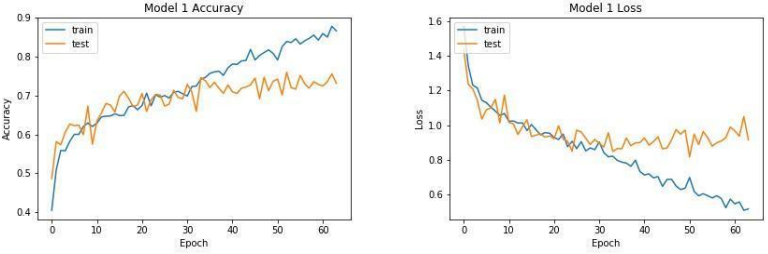
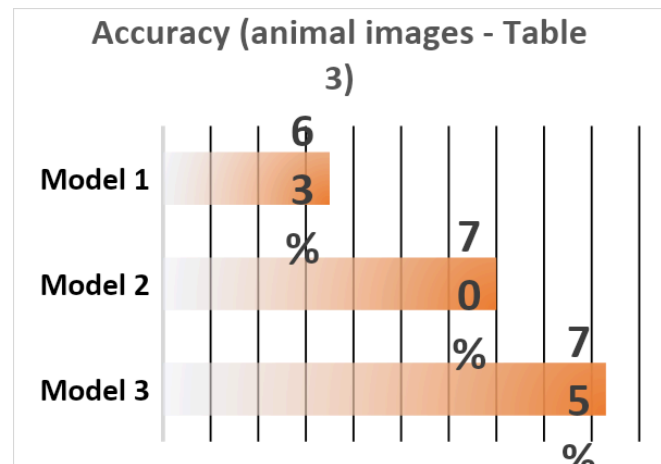
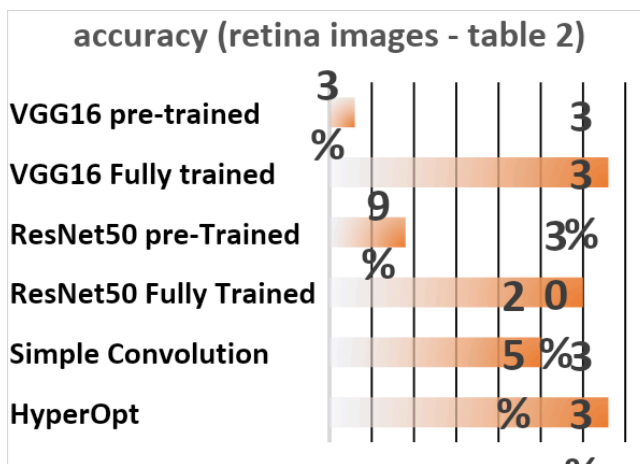
	Pooling, Flatten, 6 Dense, 1 Dropout, 1 Dense (5 classes) Activation: ReLU	and 750 for test) Balanced across classes	
--	---	--	--

Table 3: Describes the experiments run with a known Tiny ImageNet dataset with 5 animal classes

## 6. Analysis of the Results

We learned that systematic planning, followed by research helped us methodically approach this project. The biggest challenge was getting the accuracy of the model to a reasonable level. We searched for answers in various publications, articles, blogs, and GitHub repositories and did numerous experiments. Here is the summary of our observation.

1. Initial results (Table 1) with all models showed that category 0 which had 74% percent of data was the predicted category for every test image. When we balanced the dataset with an equal number of images in each of the five classes, we observed an accuracy of 25% (Table 2, #1). Adding more complexity/layers helped minimally. However, retraining VGG16 and ResNet end-to-end increased the accuracy from 25% to 30% (Table 2, #4, #5). However, accuracy with a balanced animal dataset was 74% (Table 3). This indicated that data played a key role in accuracy.
2. Pre-processing the dataset (cropping, scaling, white balancing, etc.) resulted in some improvement.
3. Hyper-Opt simplifies the search of Hyper-Parameters and provides an insight into how the loss responds to changes. The best accuracy we achieved with Hyper-Opt was 30%. This was using a VGG16+FCN model, with hyperparameters optimized by hyper-opt with preprocessing.



## 7. Conclusion

We believe Artificial Intelligence can help make accurate predictions using deep learning, especially using trained CNN models. These models can be deployed on handheld devices such as Portable Ophthalmoscope or Smartphones connected to Retinal imaging cameras, to predict the severity and progression of the DR, which can then be used by trained operators. This can potentially help early detection of Diabetic Retinopathy, which is a leading cause of vision loss around the world.



Figure 5: Image of a portable retinal imaging system. Source <sup>(8)</sup>

## 8. References

1. Diabetic Retinopathy – silently blinding millions of people worldwide  
(<http://atlas.iapb.org/vision-trends/diabetic-retinopathy/>)
2. Dataset: ( <https://www.kaggle.com/c/diabetic-retinopathy-detection/data> )
3. Tiny ImageNet: (<https://tiny-imagenet.herokuapp.com/>)

4. VGG16 ( <https://keras.io/applications/#vgg16> )
5. ResNet50 ( <https://keras.io/applications/#resnet> )
6. HyperOpt library ( <https://github.com/hyperopt/hyperopt/wiki/FMin> )
7. TensorFlow Lite (<https://www.tensorflow.org/lite>)
8. PanOptic™ Ophthalmoscope (<https://www.welchallyn.com/en/microsites/iexaminer.html>)

## 9. Acknowledgments

- Prof. Ronjon Nag, PhD, Stanford Distinguished Careers Institute, Fellow, Stanford University