Ocular Disease Recognition Using Convolutional Neural Networks

Grzegorz Meller, University of Milan

Introduction

Early ocular disease detection is an economic and effective way to prevent blindness caused by diabetes, glaucoma, cataract, age-related macular degeneration (AMD) and many other diseases. According to World Health Organization (WHO) at present at least 2.2 billion people around the world have a vision impairments, of whom at least 1 billion have a vision impairment that could have been prevented[1]. Rapid and automatic detection of diseases is critical and urgent in reducing the ophthalmologist's workload and prevents vision damage of patients. Computer vison and deep learning can automatically detect ocular diseases after providing high quality medical eye fundus images. In this paper I show different experiments and approaches towards building advanced classification model using convolutional neural networks written using TensorFlow library.

Dataset

Ocular Disease Intelligent Recognition (ODIR) is a structured ophthalmic database of 5,000 patients with age, color fundus photographs from left and right eyes and doctors' diagnostic keywords from doctors. This dataset is meant to represent ''real-life'' set of patient information collected by Shanggong Medical Technology Co., Ltd. from different hospitals/medical centers in China. In these institutions, fundus images are captured by various cameras in the market, such as Canon, Zeiss and Kowa, resulting into varied image resolutions. Annotations were labeled by trained human readers with quality control management[2]. They classify patient into eight labels including normal (N), diabetes (D), glaucoma (G), cataract (C), AMD (A), hypertension (H), myopia (M) and other diseases/abnormalities (O).

[1] WHO: World report on vision. World Health Organization (2019): https://www.who.int/publications-detail/world-report-on-vision

[2] Kaggle Ocular Disease Recognition dataset description: https://www.kaggle.com/andrewmvd/ocular-disease-recognition-odir5k

After preliminary data exploration I found the following main challenges of the ODIR dataset:

- Highly unbalanced data. Most images are classified as normal (1140 examples), while specific diseases like for example hypertension have only 100 occurrences in the dataset.
- The dataset contains multi-label diseases, because each eye can have not only one single disease but also a combination of many.
- Images labeled as "other diseases/abnormalities" (O) contains images associated to more than 10 different diseases stretching the variability to a greater extent.
- Very big and different image resolutions. Most images have sizes around 2976x2976 or 2592x1728 pixels.

All these issues take a significant toll on the accuracy and other metrics.

Data Pre-Processing

Firstly, all images are resized. At the beginning, I wanted to resize images "on the fly", using TensorFlow dataset object. Images were resized while training the model. I thought it could prevent from time-consuming images resizing at once. Unfortunately, it was not a good decision, execution of one epoch could take even 15 minutes, so I created another function to resize images before creating the TensorFlow dataset object. As a result, data are resized only once, and saved to different directory, thus I could experiment with different training approaches using much faster training execution. Initially, all images were resized to 32x32 pixels size, but quickly I realized that compressing to such a low size, even though it speeds up training process significantly, loses a lot of important image information, thus accuracy was very low. After several experiments I found that size of 250x250 pixels was the best in terms of compromising training speed and accuracy metrics, thus I kept this size on all images for all further experiments.

Secondly, images are labeled. There is a problem with images annotations in the data.csv file because the labels relate to both eyes (left and right) at once whereas each eye can have a different disease. For example, if left eye has cataract and right eye has normal fundus, the label would be cataract, not indicating diagnosis of the right eye. Fortunately, the diagnostic keywords relate to a single eye. Dataset was created in a way to provide to the model as input both left and right eye images and return overall (for both eyes) cumulated diagnosis, neglecting the fact that one eye can be healthy. In my opinion, it does not make sense from a perspective of a final user of such a model, and it is better to get predictions separately for each eye, to know for example which eye should be treated. So, I enriched the dataset by creating a mapping between the diagnostic keywords to disease labels. This way,

each eye is assigned to a proper label. Fragment of this mapping, in the form of dictionary, is presented in the Fig. 1. Label information is added by renaming image names, and more specifically, by adding to the image file name one or more letters corresponding to the specific diseases. I applied this solution, because this way I do not need to store any additional data frame with all labels. Renaming files is a very fast operation and in the official TensorFlow documentation, TensorFlow datasets are created simply from files, and label information is retrieved from the file name[3]. Moreover, some images that had annotations not related to the specific disease itself, but to the low quality of the image, like "lens dust" or "optic disk photographically invisible" are removed from the dataset as they do not pay a decisive role in determining patient's disease.

```
'abnormal pigment ': 'O',
'age-related macular degeneration': 'A',
'arteriosclerosis': 'O',
'myopia retinopathy': 'M',
'myopic maculopathy': 'M',
'suspected diabetic retinopathy': 'D',
'suspected glaucoma': 'G'
```

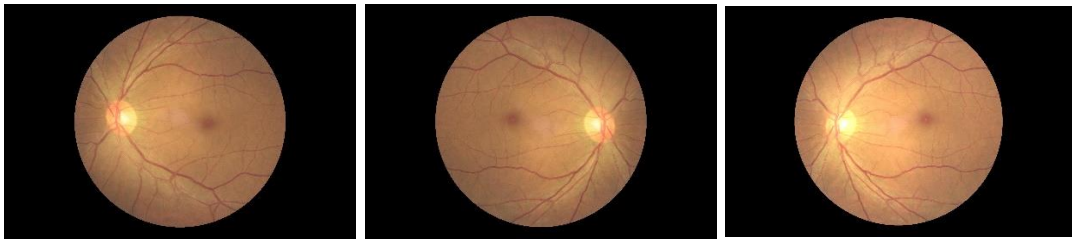
Fig. 1: Fragment of dictionary mapping specific diagnostic keyword with disease label

Thirdly, validation set is created by randomly selecting 30% of all available images. I chose 30%, because this dataset is relatively small (only 7000 images in total), but I wanted to make my validation representative enough, not to have bias when evaluating model, related to the fact, that many image variants or classes could not have their representation in the validation set. The ODIR dataset provides testing images, but unfortunately no labeling information is provided to them in data.csv file, thus I could not use available testing images to evaluate the model.

Next, data augmentation on minority classes was applied on training set to balance the dataset. Random zoom, random rotation, flip left-right, flip top-bottom were applied. At the beginning I used TensorFlow dataset object for applying data augmentation "on the fly" while training the model[4] in order to keep my solution as scalable as possible. Unfortunately, it lacks many features like random rotation, therefore I performed data augmentation before creating the TensorFlow dataset object using other libraries for image processing like OpenCV. At the beginning I also considered enhancing all images by applying contrast-limited adaptive histogram equalization (CLAHE) in order to increase visibility of local details of an image, but

[3] Loading images to TensorFlow dataset object: https://www.tensorflow.org/tutorials/load_data/images
[4] Data augmentation in TensorFlow: https://www.tensorflow.org/tutorials/images/data_augmentation

since it was adding a lot of extra noise to the images (especially to the background, which originally is black) I decided not to follow that direction. Examples of data augmentation using my function written using PIL and OpenCV libraries is presented in Fig. 2.



(a) original image (b) right-let rotation (c) random brightness

Fig. 2: Exemplary data augmentation results

Finally, the TensorFlow dataset object is created. It is developed very similarly to the one presented in official TensorFlow documentation for loading images[5]. Since the library is complicated, and not easy to use for TensorFlow beginners, I would like to share here summary of my findings on building scalable and fast input pipelines. The *tf.data* API enables you to build complex input pipelines from simple, reusable pieces. For example, the pipeline for an image model might aggregate data from files in a distributed file system. The *tf.data* API introduces a *tf.data.Dataset* abstraction that represents a sequence of elements, in which each element consists of one or more components. For example, in my image pipeline, an element is a single training example, with a pair of tensor components representing the image and its label[6]. With the idea of creating mini batches, TensorFlow introduces so called iterative learning process which is feeding to the model some portion of data (not entire dataset), training and repeating with another portion, which are called batches. Batch size informs us, how many examples will be extracted at each training step. After each step, weights are updated. I selected batch size equal to 32, in order to avoid overfitting problem, with small batch size, weights keep updating regularly and often, the downside of having small batch size is that training takes much longer than with the bigger size. One important element of *tf.data* is the ability of shuffling dataset. In shuffling, the dataset fills a buffer with elements, then randomly samples elements from this buffer, replacing the selected elements with new

[5] Loading images to TensorFlow dataset object: https://www.tensorflow.org/tutorials/load_data/images
[6] https://www.tensorflow.org/guide/data

elements[7]. It helps avoiding situation when for example images of the same class will be repetitively filled to the batch, which is not beneficial for training the model.

Building Convolutional Neural Network

In deep learning, a convolutional neural network (CNN) is a class of deep neural networks, most commonly applied to analyzing visual imagery[8]. My CNN as input takes 250x250 RGB images. The first 2D convolution layer shifts over the input image using window of the size of 5x5 pixels to extract features and save them on multi-dimensional array, in my example number of filters for first layer equals 32, so to (250, 250, 32) size cube.

After each convolution layer, rectified linear activation function (ReLU) is applied. Activation has the authority to decide if neuron needs to be activated or not measuring weighted sum. ReLU activation function is defined as $f(x) = \max(0, x)$. As we can see it is a simple calculation that returns the value provided as input directly, or the value 0.0 if the input is 0.0 or less. Because rectified linear units are nearly linear, they preserve many of the properties that make linear models easy to optimize with gradient-based methods. They also preserve many of the properties that make linear model generalize well[9].

To progressively reduce spatial size of the input representation and minimize the number of parameters and computation in the network max pooling layer is added. In short, for each region represented by the filter of a specific size, in my example it is (5, 5), it will take max value of that region and create new output matrix where each element is the max of the region in the original input.

To avoid overfitting problems, two dropouts of 45% layers were added. Several batch normalization layers were added to the model. Batch normalization is a technique for improving the speed, performance, and stability of artificial neural networks[10]. It shifts distribution of neuron output, so it better fits activation function.

Finally, the "cube" is flattened. No fully connected layers are implemented to keep simplicity of the network and keep training fast. The last layer is 8 dense, because 8 is the number of labels (diseases) present in the dataset. Since we are facing multi-label classification (data sample can belong to multiple instances) sigmoid activation function is applied to the last layer. Sigmoid function is expressed as $S(x) = \frac{1}{1+e^{-x}} = \frac{e^x}{e^x+1}$, and converts each score to the final node between 0 to 1, independent of what other scores are (in contrast to other

[7] https://www.tensorflow.org/api_docs/python/tf/data/Dataset#shuffle
[8] https://en.wikipedia.org/wiki/Convolutional_neural_network
[9] Deep Learning, Ian Goodfellow, Yoshua Bengio, and Aaron Courville.
[10] https://en.wikipedia.org/wiki/Batch_normalization

functions like, for example, softmax), thus sigmoid works best for multi-label classification problems. Since we are using sigmoid activation function, we must go with binary cross entropy loss. Selected optimizer is Adam with low learning rate of 0.0001 because of overfitting problems that I was facing during the training. Entire architecture of my CNN is presented in Fig.3.

Layer (type)	Output Shape
conv2d_4 (Conv2D)	(None, 250, 250, 32)
batch_normalization_4 (Batch	(None, 250, 250, 32)
activation_4 (Activation)	(None, 250, 250, 32)
conv2d_5 (Conv2D)	(None, 250, 250, 32)
batch_normalization_5 (Batch	(None, 250, 250, 32)
activation_5 (Activation)	(None, 250, 250, 32)
max_pooling2d_2 (MaxPooling2	(None, 83, 83, 32)
dropout_2 (Dropout)	(None, 83, 83, 32)
conv2d_6 (Conv2D)	(None, 83, 83, 128)
batch_normalization_6 (Batch	(None, 83, 83, 128)
activation_6 (Activation)	(None, 83, 83, 128)
conv2d_7 (Conv2D)	(None, 83, 83, 128)
batch_normalization_7 (Batch	(None, 83, 83, 128)
activation_7 (Activation)	(None, 83, 83, 128)
max_pooling2d_3 (MaxPooling2	(None, 27, 27, 128)
dropout_3 (Dropout)	(None, 27, 27, 128)
flatten_1 (Flatten)	(None, 93312)
dense_1 (Dense)	(None, 8)

Fig. 3: Model summary

Experiments and Results

For simplicity, I wanted to start my research with easy proof-of-concept experiments, on less challenging and smaller dataset, to test if all previous assumptions were correct. Thus, I started training simple model to detect if eye has normal fundus or cataract, training only on images labeled as N (normal) or C (cataract). The results were very satisfactory, using relatively simple network in 12 epochs my model got 93% on validation accuracy. This already shows that using CNN it is possible to automatically detect eye cataract! In each next experiment I was adding to the dataset images of another class. The fourth experiment is performed on entire ODIR dataset, achieving almost 50% validation accuracy. Results from the experiments are presented in Table 1. As we can clearly see the overall model has low results because it is hard to train it to detect diabetes correctly, since eye with diabetes look almost the same as eye with normal fundus. Detecting myopia or cataract is much easier task, because these images vary a lot from each other and from normal fundus. Illustration of different selected diseases is presented in the Fig. 4.

Class	Precision	Recall	F1 score	Accuracy
N/C	0.9328	0.9328	0.9328	0.9328
N/C/M	0.8800	0.8709	0.8754	0.8722
N/C/M/A	0.8310	0.8253	0.8281	0.8281
ALL	0.4763	0.4617	0.4689	0.4953
N/D	0.5778	0.5778	0.5778	0.5778

Table 1: Experiments results. Legend: N – normal, C- cataract, M – myopia, A – AMD, D – diabetes, ALL – model trained on entire ODIR dataset

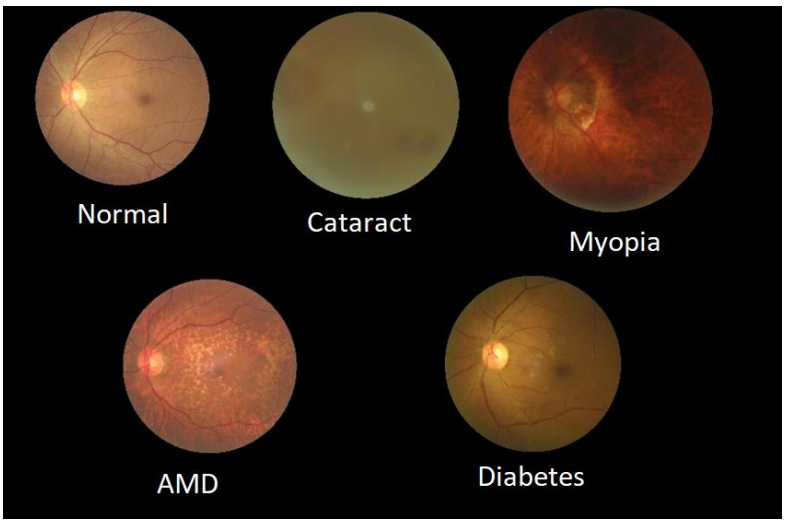


Fig. 4: Illustration of different eye diseases. Clearly Diabetes seeems to be the most challenging in detecting and cataract is the easiest as varies the most from normal fundus.

For all experiments, the same neural network architecture was used. The only difference is number of epochs each experiment needed to get to the presented results (some needed to be early stopped, other needed more epochs to learn). Also, for experiments that did not include entire dataset, softmax activation function and categorical cross-entropy loss were used, since they are multi-class not multi-label classification problems.

Applications

One of the possible applications of trained model is standardization of the output. I created script that as input takes pair of images (of the left and right eye), and provide as output cumulated result for both eyes, so it looks the same as in ODIR .csv file. Schema of this workflow is presented in Fig.5. Merging two classifications is based on taking higher results from output vectors. If there is a disease with value bigger than 0.5, automatically value for normal fundus (N) is set to 0 (because if one eye is normal, but second has disease, overall result should not point to normal fundus as prediction).

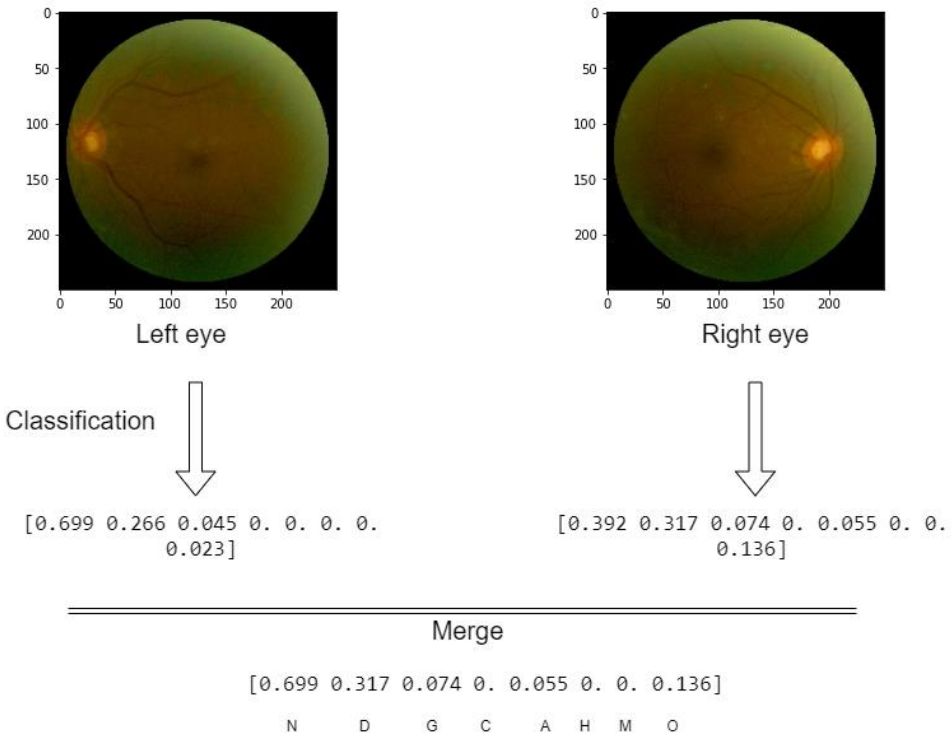


Fig. 5.

Final Considerations on Model Scalability

Nowadays, in the world of Big Data, it is crucial to evaluate every IT project, based on its scalability and reproducibility. From the beginning of implementation of this project, I put a lot of emphasis on idea, that even though it is a research project, maybe in the future with more datapoints of eye diseases the model could be re-trained, and certainly will achieve much better results having more images to train on. So, the main goal was to build universal data pipeline which is able to handle much more datapoints. This goal was mostly achieved by using advanced TensorFlow library, especially with the dataset object, that supports ETL processes (Extract, Transform, Load) on large datasets. Unfortunately, some transformations were needed to be done before creating TensorFlow dataset object, which are image resizing and augmenting minority classes. Maybe in the future it will be possible to resize images "on the fly" faster, and more augmentation functions would be added like random rotation, which was already mentioned before. But if we consider having more datapoints in the future, possibly it would not be necessary to perform any augmentations, as sufficiently enough image variations would be provided. From perspective of other popular datasets used in deep learning projects, ODIR would be considered as a small one. That is the reason why data points had to be augmented and oversampled in order to achieve sensible results.

Summary

In this project, I have proved that it is possible to detect various eye diseases using convolutional neural networks. The most satisfying result is detecting cataract with 93% accuracy. Examining all the diseases at one time, gave significantly lower results**.** With the ODIR dataset providing all important variations of specific disease to the training model was not always possible, which affects the final metrics. Although, I am sure that having bigger dataset, would increase accuracy of predictions and finally automate the process of detecting ocular diseases.

Declaration of entirely own work

I declare that this material, which I now submit for assessment, is entirely my own work and has not been taken from the work of others, save and to the extent that such work has been cited and acknowledged within the text of my. I understand that plagiarism, collusion, and copying are grave and serious offences in the university and accept the penalties that would be imposed should I engage in plagiarism, collusion or copying. This assignment, or any part of it, has not been previously submitted by me or any other person for assessment on this or any other course of study.

References

1. Source and Camera Independent Ophthalmic Disease Recognition from Fundus Image Using Neural Network. Md. Tariqul Islam, Sheikh Asif Imran, Asiful Arefeen, Mahmudul Hasan, Celia Shahnaz. Bangladesh University of Engineering and Technology.
2. The relationship between Fully Connected Layers and number of classes for the analysis of retinal images. Ajna Ram and Constantino Carlos Reyes-Aldasoro. University of London
3. Algorithms for Massive Data Lectures. Prof. Dario Malchiodi. University of Milan. https://malchiodi.di.unimi.it/teaching/AMD/2019-20/
4. TensorFlow documentation https://www.tensorflow.org/