<https://www.kaggle.com/c/diabetic-retinopathy-detection>

<https://en.wikipedia.org/wiki/Diabetic_retinopathy>

For preprocessing we both had similar approaches. There was a lot of noise in the data, so our normalization pipeline was designed to combat this.

We ended up first centering the eye, cropping out the extra blank space, downsizing, then applying brightness and contrast normalization. We trained models with varying input size - Daniel used 256x256 for the duration of the competition and Julian experimented more with different sizes (larger and smaller). We found larger input sizes to work better, but at the cost of longer time to learn and the usage of more precious GPU memory (the biggest constraint to performance).

The importance of symmetry. Given that the eye is a sphere, there are several classes of transformations we can apply that should not affect the label of the image. The major ones we used were mirroring and random rotations.

1. Normalize your data. This means removing irrelevant attributes of the input data. If brightness and contrast are not important, then normalize them out.

2. Set up your network. Start by alternating layers of 3x3 convolutions with ReLU activations and 2x2 stride 2 pooling. Each time you pool, increase the number of convolutional filters. I like to double the number of filters, others increase it linearly. Keep alternating these layers until the result is small enough to deal with in fully connected layers. Domain knowledge comes into play here, you need to know how far apart two pixels need to be before you can ignore their interaction. Stack a few fully connected ReLU or MaxOut layers on top.

3. Initialize your network with a tested (theoretically sound) method. I like sqrt(2) scaled orthogonal initialization, but Julian had good results with the Xavier method so I think either is fine. Good initialization is extremely important.

4. Train with SGD + momentum. Exploit any label-preserving transformations to artificially enlarge your dataset. Use dropout, weight decay, and weight norm penalties if necessary.

A single network following this scheme should have ended up in the top 10% on the leaderboard. Improving the result then takes some work, but the major things to try are: more convolutional layers, different activations, different numbers of filters, different pooling, different preprocessing, and other recent research (e.g. Batch Normalization). Intuition plus trial and error worked well for me in this competition.

<https://www.kaggle.com/kmader/inceptionv3-for-retinopathy-gpu-hr>

<https://www.kaggle.com/meenavyas/diabetic-retinopathy-detection>

WINNER

<https://www.kaggle.com/c/diabetic-retinopathy-detection/discussion/15801#88655>

<https://www.kaggle.com/c/diabetic-retinopathy-detection/discussion/15801#latest-370950>

<https://github.com/btgraham/SparseConvNet/tree/kaggle_Diabetic_Retinopathy_competition>

<https://github.com/facebookresearch/SparseConvNet>

<http://blog.kaggle.com/2015/09/09/diabetic-retinopathy-winners-interview-1st-place-ben-graham/> !!!!!

3th PLACE

<https://github.com/sveitser/kaggle_diabetic>

4th PLACE

<http://blog.kaggle.com/2015/08/14/diabetic-retinopathy-winners-interview-4th-place-julian-daniel/>

5th PLACE

<http://blog.kaggle.com/2015/08/10/detecting-diabetic-retinopathy-in-eye-images/>

<https://github.com/JeffreyDF/kaggle_diabetic_retinopathy>

6th PLACE

<https://www.kaggle.com/c/diabetic-retinopathy-detection/discussion/18411#latest-104787>

<https://deepsense.ai/diagnosing-diabetic-retinopathy-with-deep-learning/>

CODES

<https://github.com/benanne/kaggle-ndsb>

<https://github.com/JeffreyDF/kaggle_diabetic_retinopathy>

<https://github.com/sveitser/kaggle_diabetic>

COHEN’S KAPPA

<https://en.wikipedia.org/wiki/Cohen%27s_kappa>

<http://www.pmean.com/definitions/kappa.htm>