Kaggle Diabetic Retinopathy Detection Team o_O solution

Mathis Antony[*] Stephan Brüggemann[†]

Hong Kong, August 7, 2015

This is a competition report for the Kaggle Diabetic Retinopathy Detection competition[1].

We use convolutional neural networks trained with lasagne[2] and nolearn[3] with large color images, various types of data augmentation, dynamic resampling for class imbalance and a "per patient" feature blending strategy which takes advantage of having pairs of sample images for each patient at our disposal. The final solution is a simple average of blends with features from two deep convolutional networks and three sets of weights for each network.

1 Features Selection / Extraction

The images in this competition are too large to be conveniently trained with convolutional networks and the hardware at our disposal. We crop away all background and resize the images to squares of 128, 256 and 512 pixel. We don't apply any other image (pre)processing steps.

2 Modeling Techniques and Training

2.1 Summary of Network Architecture

The network architectures of our conv nets are shown in table 1. For nonlinearity we use leaky (0.01) rectifier units following each convolutional and fully connected (dense) layer. The networks are trained with nesterov momentum with fixed schedule over 250 epochs.

- epoch 0: 0.003

[*]sveitser@gmail.com
[†]stephan.brueggemann@arcor.de
[1]https://www.kaggle.com/c/diabetic-retinopathy-detection
[2]https://github.com/Lasagne/Lasagne
[3]https://github.com/dnouri/nolearn

			net A			net B		
		units	filter	stride	size	filter	stride	size
1	Input				448	4		448
2	Conv	32	5	2	224	4	2	224
3	Conv	32	3		224	4		225
4	MaxPool		3	2	111	3	2	112
5	Conv	64	5	2	56	4	2	56
6	Conv	64	3		56	4		57
7	Conv	64	3		56	4		56
8	MaxPool		3	2	27	3	2	27
9	Conv	128	3		27	4		28
10	Conv	128	3		27	4		27
11	Conv	128	3		27	4		28
12	MaxPool		3	2	13	3	2	13
13	Conv	256	3		13	4		14
14	Conv	256	3		13	4		13
15	Conv	256	3		13	4		14
16	MaxPool		3	2	6	3	2	6
17	Conv	512	3		6	4		5
18	Conv	512	3		6	n/a		n/a
19	RMSPool		3	3	2	4	2	2
20	Dropout							
21	Dense	1024						
22	Maxout	512						
23	Dropout							
24	Dense	1024						
25	Maxout	512						

Table 1: Convolutional network architectures.

- epoch 150: 0.0003
- epoch 220: 0.00003

For the nets for 256 and 128 pixel images we stop training already after 200 epochs. We apply L2 weight decay with factor 0.0005 to all layers. We treat the problem as a regression problem with mean squared error objective and threshold at $(0.5, 1.5, 2.5, 3.5)$ to obtain integer levels for computing the kappa scores and making submissions. The convolutional networks have untied biases.

2.2 Resampling

The classes in the dataset are highly imbalanced. We found the following resampling strategy to work well in practice. Initially we sample from all classes such that all classes are represented equally on average. We then gradually decrease oversampling of rare classes. Let $\mathbf{w_0}$ be the initial resampling weights and $\mathbf{w_f}$ the final resampling weights. The resampling weights for level 0 to 4 at epoch t are given by

$$\mathbf{w}_i = r^{t-1}\mathbf{w}_0 + (1 - r^{t-1})\mathbf{w}_f \tag{1}$$

where we set $r = 0.975$, $\mathbf{w}_0 \approx (1.36, 14.4, 6.64, 40.2, 49.6)$ and $\mathbf{w}_f = (1, 2, 2, 2, 2)$ which were values we found to work well for initial convergence and final score. We think it is likely that similar or better results could be achieved by using a dynamically weighted objective function but being still unfamiliar with the theano library we decided to opt for resampling instead as it was easier to implement.

2.3 Initialization and "Pretraining" of Smaller Architectures

We didn't succeed in training the large convolutional networks from scratch. We first train smaller networks on 128 pixel images, then we use the trained weights to (partially) initialize networks of intermediate size which are then trained on 256 pixel images. Finally we repeat this procedure for the final networks that and train them on 512 pixel images. We use orthogonal[4] initialization for weights and constants for biases.

2.4 Data Augmentation

We apply translation, stretching, rotation, flipping as well as color augmentation at all times. We also scale and center each image channel (RGB) to have zero mean and unit variance over the training set. The output sizes for the data augmentation pipeline are 112/224/448 pixel for the 128/256/512 pixel input images.

2.5 Feature Extraction

We extract features from the last pooling layer of the convolutional networks. To increase the quality of the extracted features, we repeat the feature extraction up to 50 times (with different augmentations) per image and use the mean and standard deviation of each feature as input to our blending network.

2.6 Per Patient Blend

We extract mean (μ) and standard deviation (σ) of the RMSPool layer output for 50 pseudo random augmentations for three sets of weights (best validation score, best kappa, final weights) for net A and B. For each eye (or patient) use the following as input features for blending,

$$\mathbf{x} = (\mu_{\text{this eye}}, \mu_{\text{other eye}}, \sigma_{\text{this eye}}, \sigma_{\text{other eye}}, \delta_{\text{right}})$$

where $\delta_{\text{right}} \in \{0, 1\}$ is an indicator variable for right eyes. We standardize all features to have zero mean and unit variance and use them to train a simple fully connected network with the architecture shown in table 2. The training procedure for the blend network is summarized below.

[4] https://github.com/Lasagne/Lasagne/blob/master/lasagne/init.py#L329-L359

Input	8193
Dense	32
Maxout	16
Dense	32
Maxout	16

Table 2: Blend network architecture.

- Rectifier nonlinearities after each fully connected (dense) layer.
- L1 regularization with factor 2e-5 is applied to the first layer and L2 regularization with factor 0.005 is applied on every layer.
- Adam[5] updates with fixed learning rate schedule over 100 epochs.
  - epoch 0: 5e-4
  - epoch 60: 5e-5
  - epoch 80: 5e-6
  - epoch 90: 5e-7
- Mean squared error objective.
- Batches of size 128, replace batch with probability
  - 0.2 with batch sampled such that classes are balanced
  - 0.5 with batch sampled uniformly from all images (shuffled)

The score of our final ensemble without per patient blend is 0.824 on the private leaderboard. With per patient blend that score increases to 0.845.

3 Generating our Solution

Please refer to the README.md and make_kaggle_solution.sh files in the code repository.

4 Additional Comments and Observations

The data set for this competition is special in that the images contain important features such as Micro-aneurysms which are very small with respect to the size of the retina. We've learned that it's important for performance to use rather large input images. On our 10% validation set, our convolutional neural networks achieve kappa scores of 0.70 for 256 pixel images and 0.80 for 512 pixel images. We also tried 768 pixel images and this further increased kappa to 0.81. At this point the data augmentation became computationally very demanding and as we didn't see

[5]http://arxiv.org/abs/1412.6980

a significant improvement after blending per patient features going from 512 pixel to 768 pixel images we stuck with the latter.

It is considerably easier to train the large convolutional networks when the leaky rate of the rectifiers is increased. A per patient blend of features extracted over 50 pseudo random augmentations would achieve a validation kappa score of about 0.84. When we switched to using very leaky rectifiers (with leaky rate 0.33) and trained the large network from scratch that the score would decrease to about 0.83. The difference is rather small but significant for competition results. We were thrown off track for quite a while because that drop in performance would only manifest itself after blending, but not during training where the validation kappas for both types of rectifier units were very similar. After ruling out any other factors, we concluded that either the change in rectifier units or the change in training procedure (initialization from smaller nets versus training from scratch) caused the decrease in performance. Towards the end of the competition we simply went back to using 0.01 leaky rectifiers everywhere.

5 Simple Features and Methods

Our final solution is an average of six per patient blends for the two convolutional network architectures and three different set of trained weights. Ensembling only increased the private leaderboard score from 0.840 to 0.845 and one would have to decide if this small performance increase justifies training a second network and extracting features for six times as many augmentations of each input image.

We also noticed that blending the features for both patient eyes leads to significant performance improvements for various types of architectures. For this particular problem it provides a simple way to improve performance without having to redo or modify the feature extraction part of the pipeline.

6 Acknowledgments

We want to thank kaggle, the California Healthcare Foundation and EyePACS for putting the competition together. We also thank the winners[6][7] of the national data science bowl competition for their code and reports which heavily influenced our solution. Last but not least we want to thank the open source community and especially the contributors to lasagne and nolearn for all the awesome software and documentation which enabled us to compete in this way.

[6] https://github.com/benanne/kaggle-ndsb

[7] https://www.kaggle.com/c/datasciencebowl/forums/t/13166/happy-lantern-festival-report-and-code