Summary of our Solution to the Kaggle Diabetic Retinopathy Detection Competition

Jun Xu, John Dunavent, Raghu Kainkaryam (Team Reformed Gamblers, 3rd place)

Summary

Our solution was an ensemble of 9 convolutional neural networks. We used a variety of model architectures. Our best performing were variations of Simonyan and Zisserman, closely followed by models featuring the fractional max pooling layers developed by the 1st place finisher (Graham). Also included was a model utilizing cyclic pooling as described in the winning solution to the National Data Science Bowl (Dieleman). We found that using large image sizes and combining information from both eyes were key to getting good performance.

Model Details

We used the Torch and cuDNN libraries for all of our work. We utilized several different CUDA GPUs (K80, 980GTX, Titan Black).

We trained models on a variety of image sizes ranging from 384x384 to 1024x1024 pixels. Our largest models were trained using data parallelism across multiple GPUs using code based on Soumith's Imagenet example code.

Most of our models were trained using a clipped MSE error function. This was a standard MSE objective with the model output clipped to [0, 4] prior to evaluating. This allowed a model to push predictions below 0 for training images with ratings of 0 without generating any error signal. Similarly, images with ratings of 4 could get predictions much greater than 4 without any error. We hypothesized that clipping the scores this way increased model output range, making it easier to separate the classes.

We also used a staircase-like loss function formed by combining 4 sigmoid functions. Performance between the 2 objectives was similar.

The following details were shared by all of our models:

1. Predictions were generated one eye at a time
2. Data augmentation via random affine transformation. We used the WarpAffine function from Nvidia's NPP library to perform the transformations on GPU. This resulted in increased GPU memory usage, but was faster than performing the operations on the CPU. In a typical setup, images were randomly cropped to 85-95%, horizontally flipped, rotated between 0 and 360 degrees and then scaled to the desired model input size
3. Channel-wise global contrast normalization was applied to normalize image color
4. PreLU weight initialization. We found this initialization method helped train models with large numbers of layers
5. Leaky rectifier non-linearities (0.1 leak factor)
6. Nesterov momentum of 0.9

Throughout the competition, we found it necessary to initialize training at a very low learning rate (10^{-4} or 10^{-5}) for several epochs to stabilize the model weights. After this 3-5 epoch slow-start, we typically used a learning rate schedule of 0.003 for 100 epochs, 0.001 for 30 epochs and 0.0001 for 20 epochs.

Table 1 shows architecture details for select models.

Post-Processing

We sought to exploit the fact that each patient, in both training and test datasets, had left and right eye images. We used a simple linear model to pool the predictions from both eyes. We reused the 10% early stopping validation set to fit this model. We made several unsuccessful attempts to train a neural network architecture that used both eyes as input.

Submissions were required to be integer-valued between 0 and 4. Rather than simple rounding, we performed a grid search of possible cutoffs in a 10% sample and selected the combination leading to integer values with the best quadratic weighted kappa score.

Ensemble Details

We combined predictions from our 9 convnet models using a L1/L2 regularized linear regression model. The input features were the integer-valued predictions after left-right pooling and cutoff optimization. The continuous ensemble predictions were converted to integer values as before.

Discussion

We believe our solution for combining data from both eyes in each prediction was not as effective as the 1st and 2nd place team solutions. We combined predictions using a linear model on a 10% holdout set. We thought this would minimize overfitting. The other winning teams used more expressive models (neural networks, random forests) and fit them using the entire training set.

While left-right pooling and optimizing the cutoffs delivered noticable performance gains, ensembling different models led to only modest improvement.

Input size was a key driver of performance. Using the same architecture, doubling the image size from 385x385 to 767x767 improved our validation set kappa scores from 0.808 to 0.843. However, we gained no additional performance by further size increases. We spent a lot of time trying to fit models to very large image sizes (bigger than 1024x1024) using pretrained intermediate layers. These models were highly overfit and not included in our final submissions.

We'd like to thank the California Healthcare Foundation and EyePACs for sponsoring and providing data for the competition. Many thanks to the Kaggle community. We leaned heavily on work shared by past competition winners. Finally, many thanks to the Torch community and to Soumith Chintala for sharing the Imagenet sample code which we adapted for this competition.

Model ID	Image Size	Validation Kappa	Objective	Architecture	Comments
cyc28	480x480	0.784	Clipped MSE	C8-3, C8-3, C8-3, MP4-4, CS, C32-3, C16-3, MP2-2, CR C64-3, C32-3, MP2-2, CR C128-3, C128-3, C64-3, MP2-2, CR C256-3, C256-3, C128-3, MP2-2, CR C512-3, C512-3, CP D(0.5), FC1024, FC1024	This was one of our earlier models. Lower layers were initialized using an earlier model. Weak performance is probably driven by the 4x4 maxpool layer and the small number of feature maps in the lower layers
m41	385x385	0.808	Clipped MSE	C32-4, C32-4, MP3-2, C64-4, C64-4, MP3-2 C128-4, C128-4, MP3-2, C256-4, C256-4, MP3-2 C384-4, C384-4, MP3-2, C512-4, C512-4, MP3-2 C512-4, C512-4, MP3-2, D(0.5), FC1024, FC1024	Testing 4x4 filter size
m42	385x385	0.809	Clipped MSE	C16-3, C16-3, C32-3, C32-1, MP3-2 C64-3, C64-3, MP3-2, C128-3, C128-3, MP3-2 C256-3, C256-3, MP3-2, C384-3, C384-3, MP3-2 C512-3, C512-3, MP3-2, C512-3, C512-3, MP3-2 D(0.5), FC1024, FC1024	
m46	767x767	0.843	Clipped MSE	C16-3, C16-1, MP3-2, C32-3, C32-1, MP3-2 C64-3, C64-1, MP3-2, C128-3, C128-1, MP3-2 C256-3, C256-1, MP3-2, C384-3, C384-1, MP3-2 C512-3, C512-3, MP3-2, C512-3, C512-3, MP3-2 D(0.5), FC1024, FC1024	The remaining models in our ensemble were slight variations on this model
m47	724x724	0.835	Clipped MSE	C16-3, FMP1.414, C16-3, FMP1.414 C32-3, FMP1.414, C32-3, FMP1.414 C64-3, FMP1.414, C64-3, FMP1.414 C128-3, FMP1.414, C128-3, FMP1.414 C192-3, FMP1.414, C192-3, FMP1.414 C256-3, FMP1.414, C256-3, FMP1.414 C384-3, FMP1.414, C384-3, FMP1.414 C512-3, FMP1.414, C512-3, FMP1.414 D(0.5), FC1024, FC1024	Pseudorandom fractional max pooling $\alpha = 1.414$

Table 1. Architecture Details for Select Models

All models used leaky ReLU non-linearities following each weight layer

C*n-s*: convolutional layer with *n* feature maps and filter size *s*

MP*s-r: s*x*s* max-pooling layer with stride *r*

D(0.5): 50% dropout layer

FC1024: fully-connected layer with 1024 weights

CS, CR, CP: Cyclic slice, rolling and pooling layers (Dieleman)

FMP*α:* pseudorandom fractional max-pooling layer. Output layer is reduced by factor *α* (Graham)