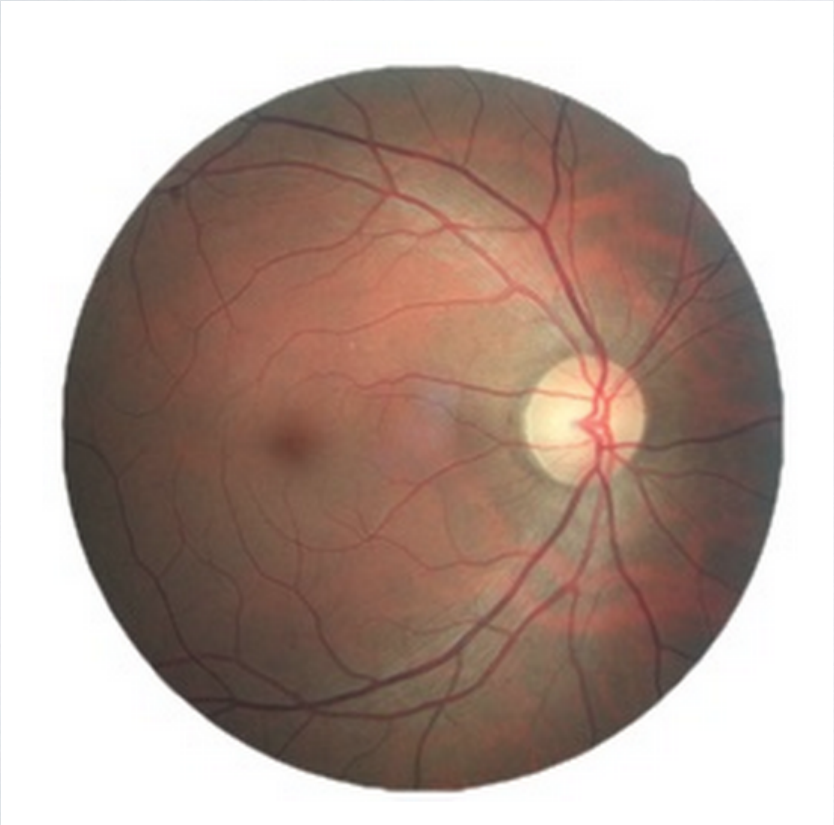
Retinopathy Net



Alberto Benavides
Robert Dadashi
Neel Vadoothker

Motivation

- We were interested in applying deep learning techniques to the field of medical imaging
- Field holds a lot of promise and can help save lives
- Kaggle is currently holding a competition to build software that can automatically detect different stages of the eye disease Diabetic Retinopathy using high quality medical images

What is Diabetic Retinopathy?

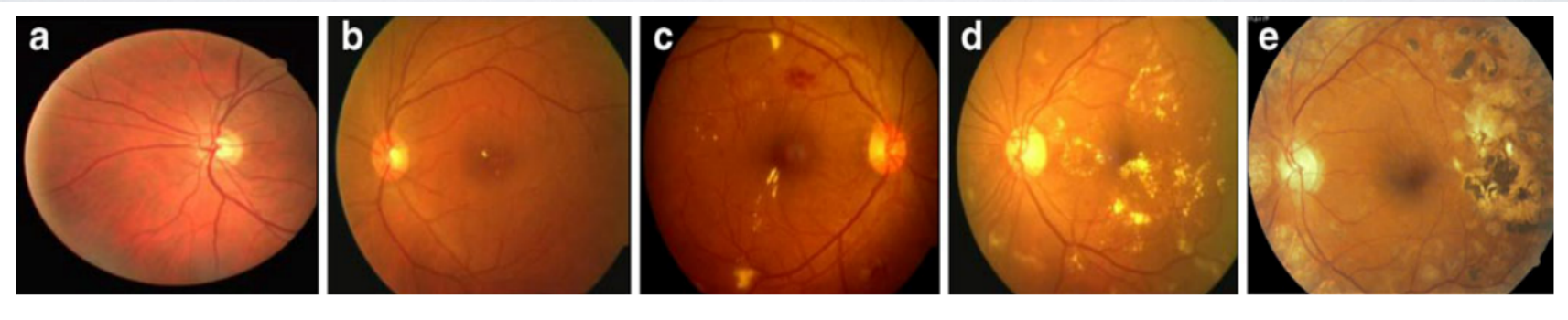
- Leading cause of blindness in the developed world, affecting over 93 million people
- Usually associated with long term diabetes in patients
- Progress of disease can be slowed if caught in time, but often doesn't show symptoms until it's too late
- Diagnosing this eye condition is a time consuming process for trained clinicians

Normal Vision

Moderate DR Vision

Different Stages of DR

Healthy ⟷ Unhealthy



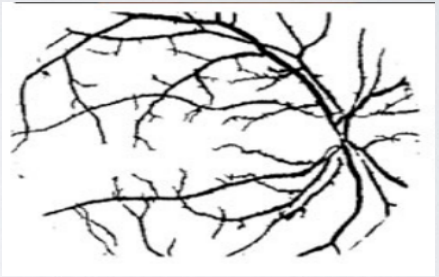
Symptoms

- Blood Vessels
- Micro-aneurysms
- Exudates
- Hemorrhage

Blood Vessels

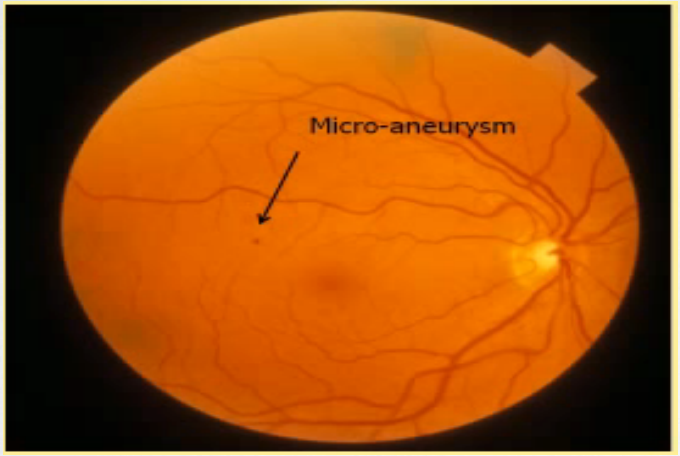
- First major symptom are the blood vessels
- Eyes with DR have damaged blood vessels that are unable to nourish the retina
- See right for examples

Normal Blood Vessels



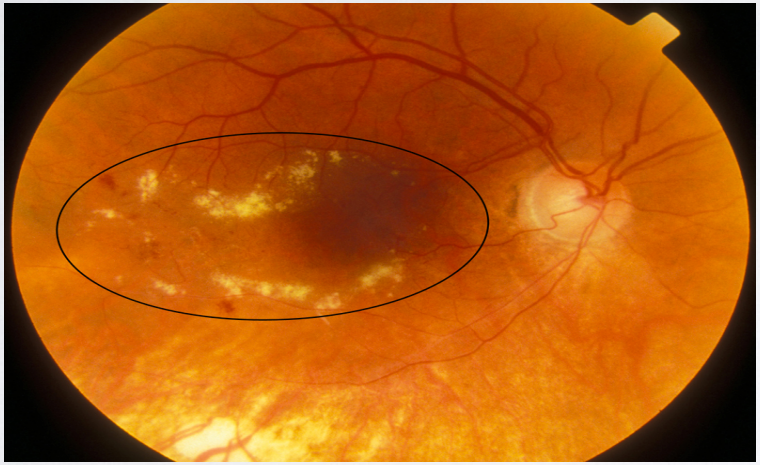
Damaged Blood Vessels

Micro-Aneurysms



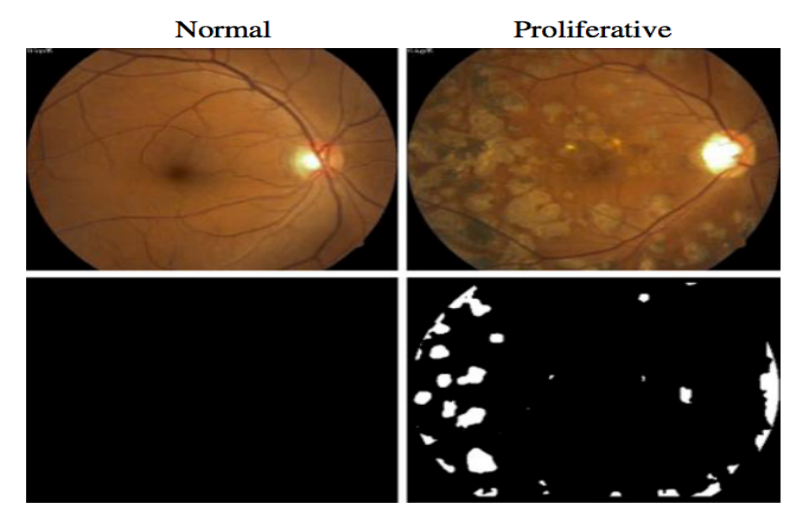
- Tiny area of blood protruding from an artery or vein in the back of the eye
- This is due to damaged blood vessels eventually blowing up and leaking blood

Exudates



- In addition to blood, sometimes fats spill out from the damaged blood vessels
- These fats harden into weird yellow spots on the retina

Hemorrhages



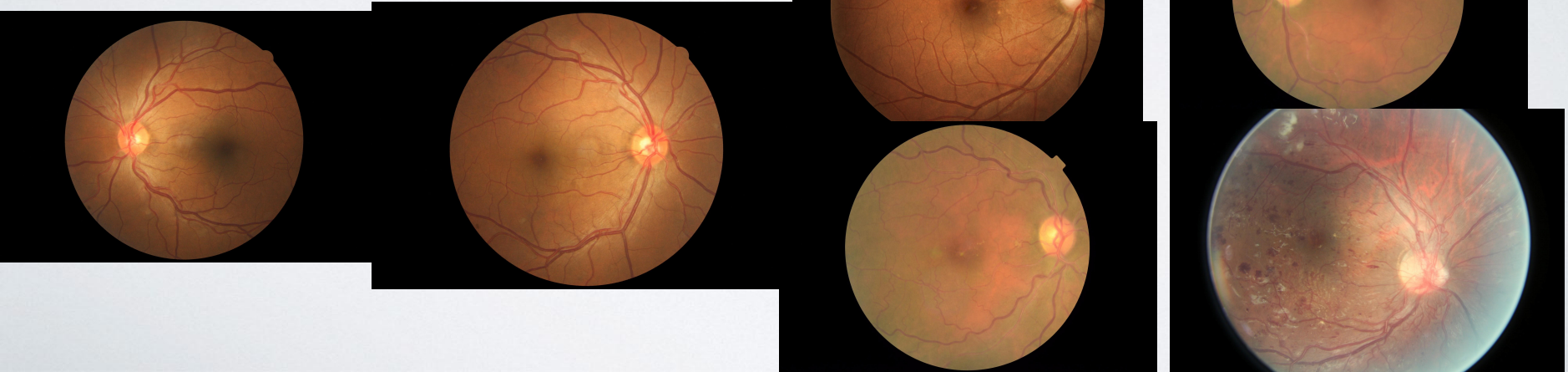
- As advanced DR occurs, the blood vessels become really leaky and hemorrhages occur in the eye
- Visually, hemorrhages are characterized by dark spots on the eye

Data

- Kaggle provides a large training dataset of approximately 35,000 medical images
- Each image is of extremely high quality, but the images aren't standardized.
- About 40GB worth of training data! About 55GB of testing data.
- Each image is labeled from 0-4 where 0 is healthy and 4 is proliferative DR

Data

- Data comes as both left eye and right eye
- Has varying quality
  - Some images are poorly lit

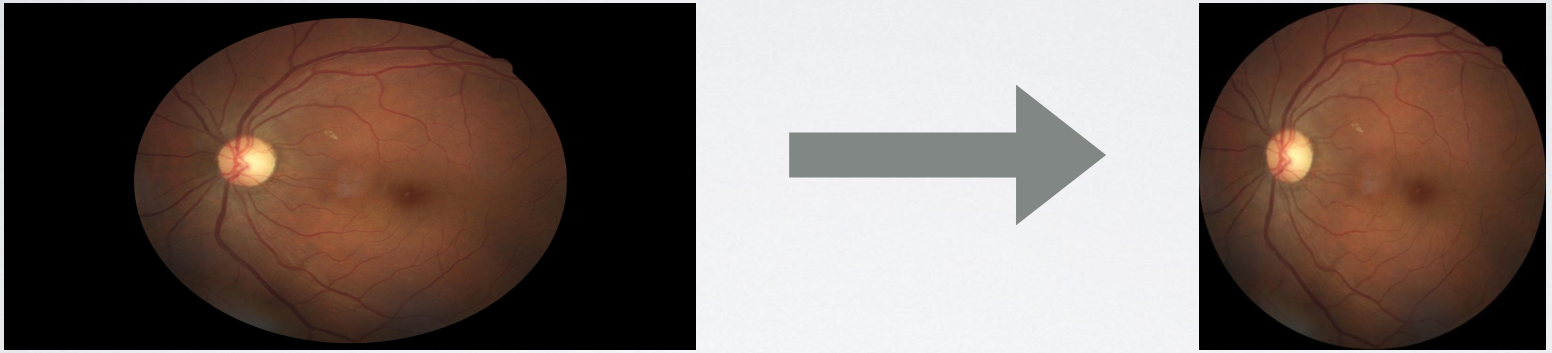


Outline of Our Solution

- Preprocessing steps
- Class Imbalance Problem
- Convergence tweaks
- Architecture
- Regularization
- Data Augmentation

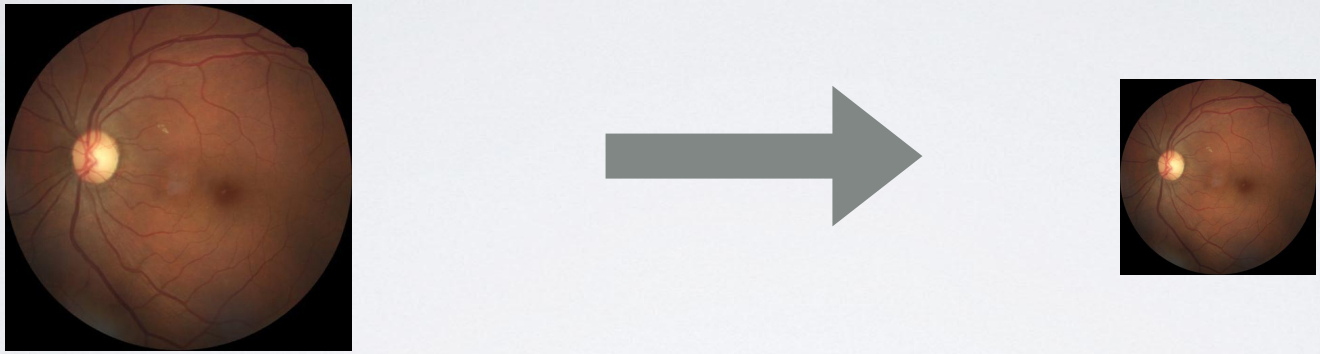
Preprocessing Steps (1)

1) Crop out black



Preprocessing Steps (2)

2) Resize images while preserving aspect ratio



Try multiple image sizes including 512x512, 128x128, and 32x32

Class Imbalance Problem

- A major problem we had in our data set is that the number of images from each class is heavily skewed
- About 28,000 images for class 0, 3000 for class 1, 2000 for class 2, 800 for class 3, and only 700 for class 4
- 3 approaches we are taking:
    - Adjust cost function to penalize misclassifications in proportion to frequency in training data
    - Force minibatches to have equal number of observations from each class
    - Use data augmentation to balance classes by "creating" new examples

For illustration purposes:

$$\frac{e^{B_i X}}{\sum_{j=1}^{K} e^{B_j X}}$$

$$\frac{p(i) \times e^{B_i X}}{\sum_{j=1}^{K} p(j) \times e^{B_j X}}$$

Convergence Tweaks (1)

- In addition to SGD, we also implemented Momentum
- On MNIST dataset, Momentum empirically increases converged by a factor of 2-3 times!
- Momentum borrows from physics concepts. Basically, the gradient descent procedure builds momentum towards a direction if the gradient stays in the same direction from iteration to iteration.

```
# Momentum update
v = mu * v - learning_rate * dx # integrate velocity
x += v # integrate position
```

Convergence Tweaks (2)

- We found some difficulty in getting our net to converge due to learning rate parameter
- To help remedy this, we implemented RMSProp which is an adaptive learning rate procedure
- Basically, it speeds up learning as long as performance is improving, and slows down learning when performance is not

```
cache = decay_rate * cache + (1 - decay_rate) * dx**2
x += - learning_rate * dx / np.sqrt(cache + 1e-8)
```

Convergence Tweaks (3)

- We also combined RMSProp with Momentum
  - Empirically, this didn't result in faster convergence
- Finally, we also tried Adadelta
  - Basically an adaptation of RMSProp in that it also uses historical gradients
  - The paper is at: http://arxiv.org/abs/1212.5701
- Adadelta has an advantage in that it doesn't require any hyperparameters like a learning rate or momentum
- Ended up using adadelta for the time being

Algorithm 1 Computing ADADELTA update at time t

Require: Decay rate ρ, Constant ϵ

Require: Initial parameter x_1

1: Initialize accumulation variables $E[g^2]_0 = 0$, $E[\Delta x^2]_0 = 0$

2: **for** $t = 1 : T$ **do** %% Loop over # of updates

3: Compute Gradient: g_t

4: Accumulate Gradient: $E[g^2]_t = \rho E[g^2]_{t-1} + (1-\rho)g_t^2$

5: Compute Update: $\Delta x_t = -\frac{\text{RMS}[\Delta x]_{t-1}}{\text{RMS}[g]_t} g_t$

6: Accumulate Updates: $E[\Delta x^2]_t = \rho E[\Delta x^2]_{t-1} + (1-\rho)\Delta x_t^2$

7: Apply Update: $x_{t+1} = x_t + \Delta x_t$

8: **end for**

Architecture

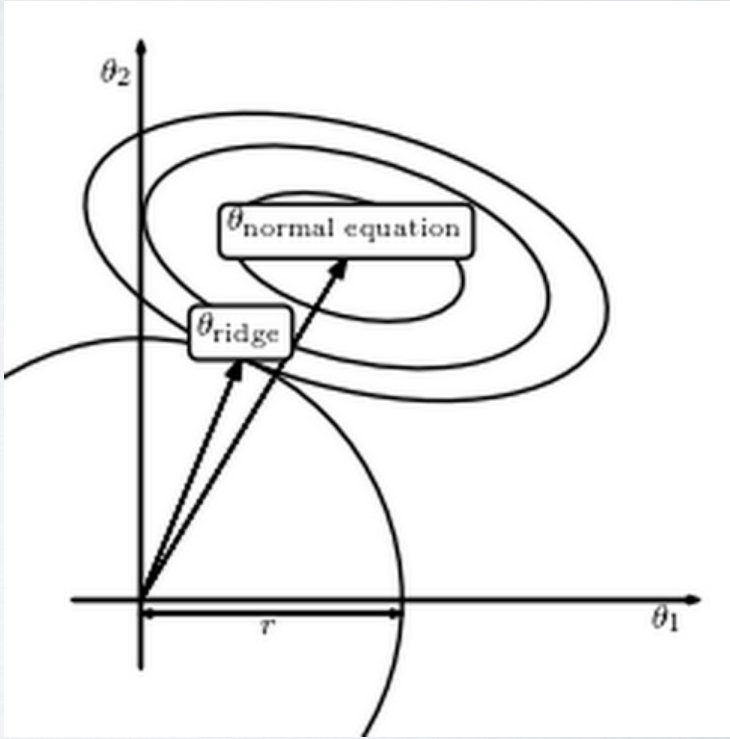
We tried two distinct architectures.

1) The basic Lenet5 architecture with some adaptations for our problem set, including the usage of color images
2) Another custom architecture:
   a) 2 convolution layers with 2x2 max pooling
   b) another convolution layer with no max pooling
   c) a fourth convolution layer with 2x2 max pooling
   d) 3 fully connected layers with leaky rectifiers
   e) a final classifer layer (logistic regression)

Regularization(1)

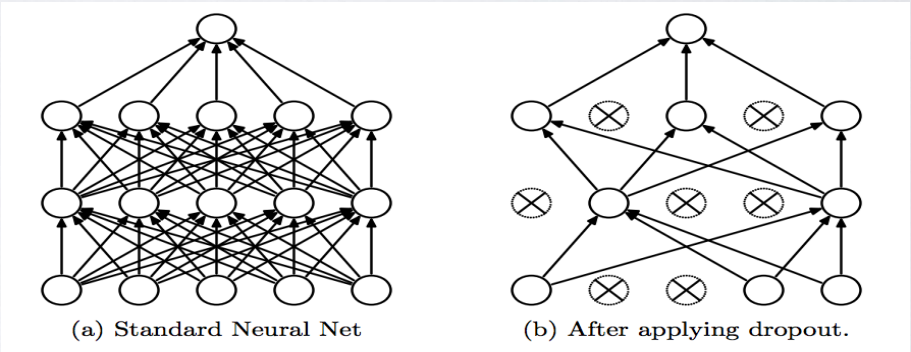
- Tried L2 regularization on all weight parameters to help combat overfitting
- Set the same weight across all layers
- Easy to prevent overfitting but did not help predictive performance much

$$F(x) = f(x) + \lambda \|x\|_2^2$$



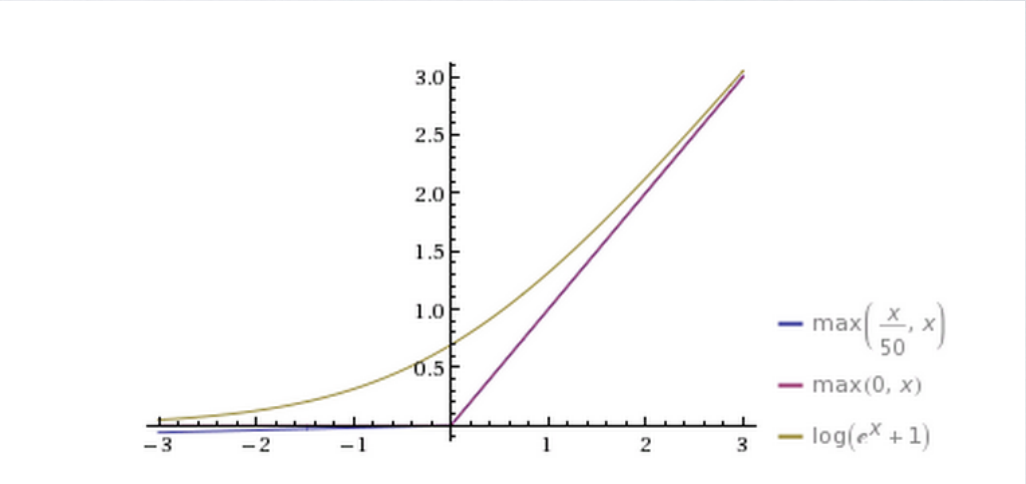
Regularization(2)

- Also implemented Dropout on each layer in our net
- Basic idea is to zero out neurons at random during training
- This helps regularize the net because neurons are forced to be less codependent and thus learn useful discriminative features



Regularization(3)

- Also used leaky rectifiers (RELUs) in the fully connected (hidden) layers
- Essentially, when x < 0, leaky RELUs have a small slope instead of zero
- Seemed to help keep overfitting down a lot
- Experimented with values between 0 and .5
    - Ended up using values nearer .5

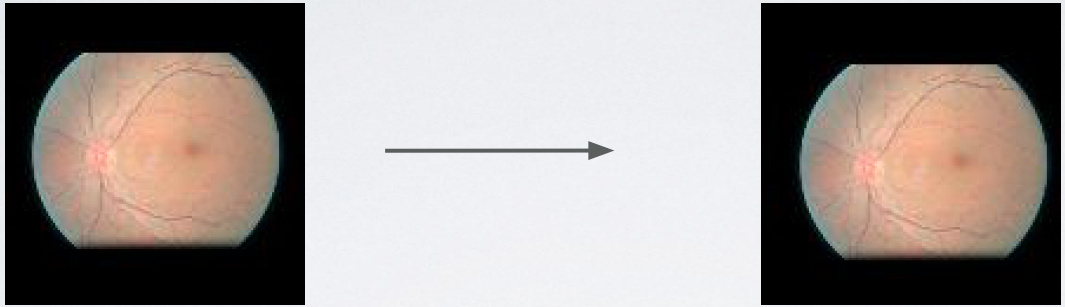


Data Augmentation

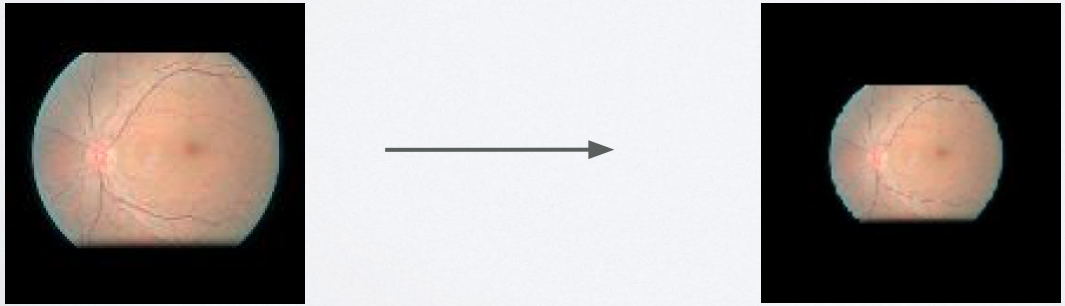
- More data -> prevent from overfitting
- From each image: apply a series of random transformations

Data Augmentation

Translation:

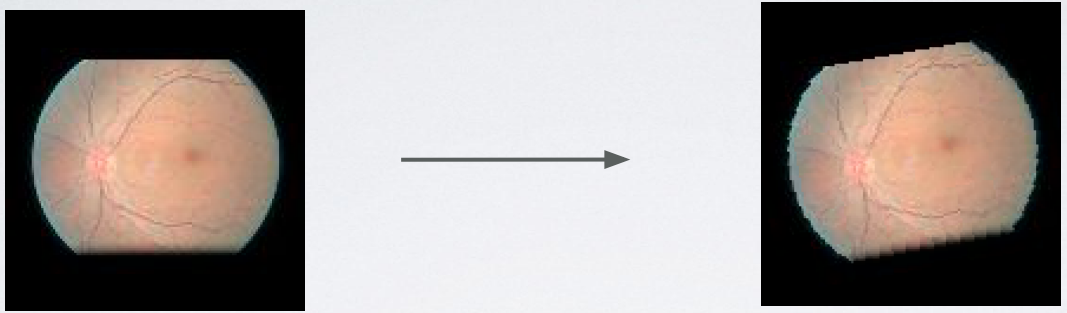


Zooming:

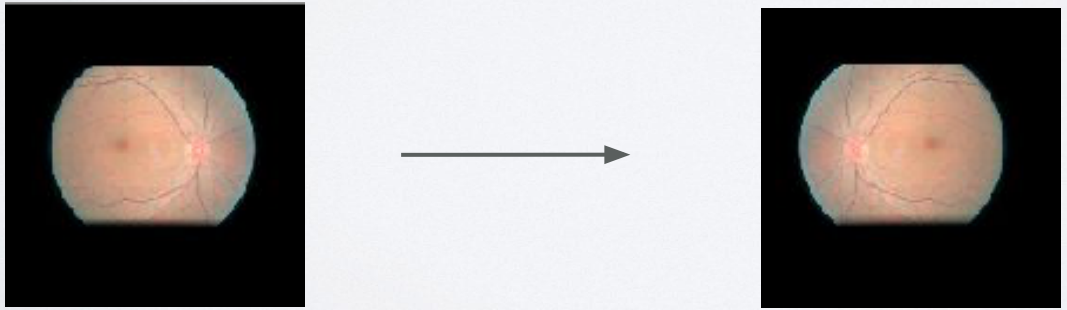


Data Augmentation

Rotation:

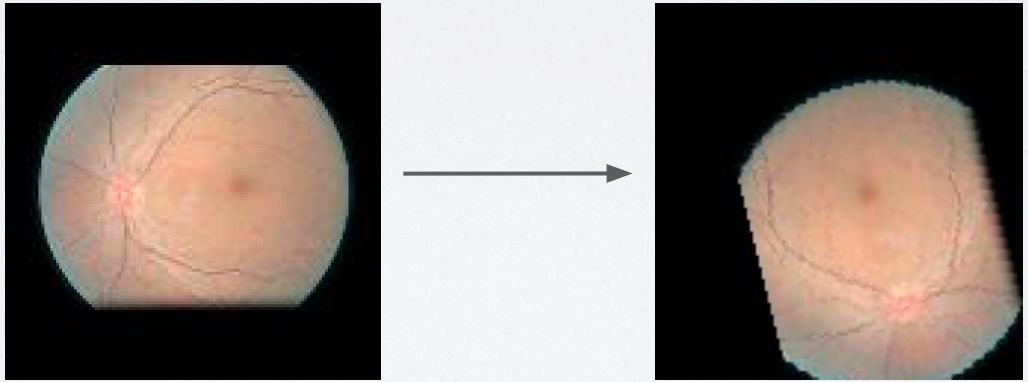


Flipping:



Data Augmentation

Combining all the random perturbations:



Limitations

- Using Amazon GPUs
  - Costs $.65/hour for on demand instances
- Dataset is very large
  - Costs $.10/GB/month
  - Train set is approximately 40 GB
  - Test set is approximately 55 GB
- Models take a long time to run
  - Approximately 6 to 8 hours for only 200 epochs
  - Sometimes models can take days to train
  - Takes a long time to tell if changes are working well
  - Many hyperparameters to optimize

Results

- We have achieved 33% validation accuracy