Machine Learning Engineer Nanodegree

Capstone Project

Winfred Selwyn Ooh

March 27th 2018

Project Overview

Diabetic Retinopathy screening with a robust machine learning modelon datasets of retinal fundus photographs.

Human clinicians identify Diabetic Retinopathy (**DR**) with the presence of lesions associated with vascular abnormalities. If left untreated, the disease leads to the patient’s blindness. With an increasing segment of the population suffering from diabetes worldwide, an effective method for screening and early diagnosis is required to avoid burdening available medical infrastructure.

A large dataset of colour retinal fundus photographs is provided by **EyePACS** for this project.

Link to project dataset: https://www.kaggle.com/c/diabetic-retinopathy-detection/data

This project will be

Additional reading:

- InceptionV1 variant of GoogleNet: https://arxiv.org/pdf/1502.03167.pdf
- Inception-V3 was proposed by Szegedy et al. : https://arxiv.org/pdf/1409.4842.pdf

Problem Statement

With the steady increase of individuals suffering from diabetes (in 2008, 180 million people worldwide, estimated to go up to 360 million by 2030 - World Health Organization), detecting DR places a burden on existing medical infrastructure.

Detecting DR manually is a time-consuming process requiring a trained clinician to examine and evaluate digital colour fundus photographs of the retina.

Treatment is usually delayed as patients suffering from deteriorating eyesight and DR are unaware they have diabetes.

With deep learning models, the process of diagnosis can be largely automated.

(*source: kdnuggets.com*) The general applicability of neural networks is one of their advantages, but this advantage turns into a liability when dealing with images. The **convolutional neural network** make a conscious tradeoff: the network is designed for specifically handling the images, some generalization is sacrificed for feasible solution.

The clinician should be able to feed photos into the models and get immediate feedback on the patient’s DR level of severity, ranging on a scale of 0 – 4.

Metrics

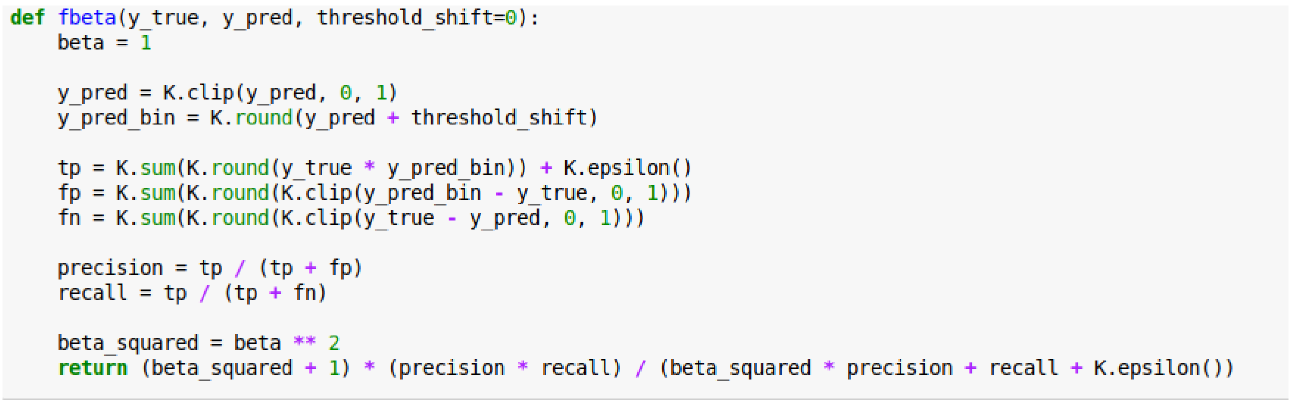
The trained neural network generates a continuous number between 0 and 4 for referable diabetic retinopathy and other diabetic retinopathy classifications, corresponding to the probability of that condition being present in the image.

Initially, during the proposal phase of the project, the proposed metrics was a **Kappa Score**.

The final metric used to measure this project’s success will be an **F1 Score.** A team from **Google Research** released a paper on DR Detection with Tensorflow and measured the accuracy of the **InceptionV3** model based on an F1 score.

$$F1 = 2\frac{PR}{P+R}$$

Expressed with code:



```
def fbeta(y_true, y_pred, threshold_shift=0):
    beta = 1

    y_pred = K.clip(y_pred, 0, 1)
    y_pred_bin = K.round(y_pred + threshold_shift)

    tp = K.sum(K.round(y_true * y_pred_bin)) + K.epsilon()
    fp = K.sum(K.round(K.clip(y_pred_bin - y_true, 0, 1)))
    fn = K.sum(K.round(K.clip(y_true - y_pred, 0, 1)))

    precision = tp / (tp + fp)
    recall = tp / (tp + fn)

    beta_squared = beta ** 2
    return (beta_squared + 1) * (precision * recall) / (beta_squared * precision + recall + K.epsilon())
```

Links:

- Video: https://www.youtube.com/watch?v=oOeZ7IgEN4o
- Journal : https://jamanetwork.com/journals/jama/fullarticle/2588763
- Blog Post : https://research.googleblog.com/2016/11/deep-learning-for-detection-of-diabetic.html

A benchmark vanilla model with 4 layers will be used to provide a benchmark F1 and accuracy score.
The final model will built on **InceptionV3**, which is an approach similar to that taken by the Google Research team.

Data Exploration

(*please visit Exploratory Data Analysis.ipynb*)

A dataset of 32,796 images have been provided by EyePAC on Kaggle for training, divided into the following classes :

- Class 0, no diabetic retinopathy: 21978
- Class 1, mild diabetic retinopathy: 2078
- Class 2, moderate diabetic retinopathy: 4568
- Class 3, severe diabetic retinopathy: 770
- Class 4, proliferative diabetic retinopathy: 606

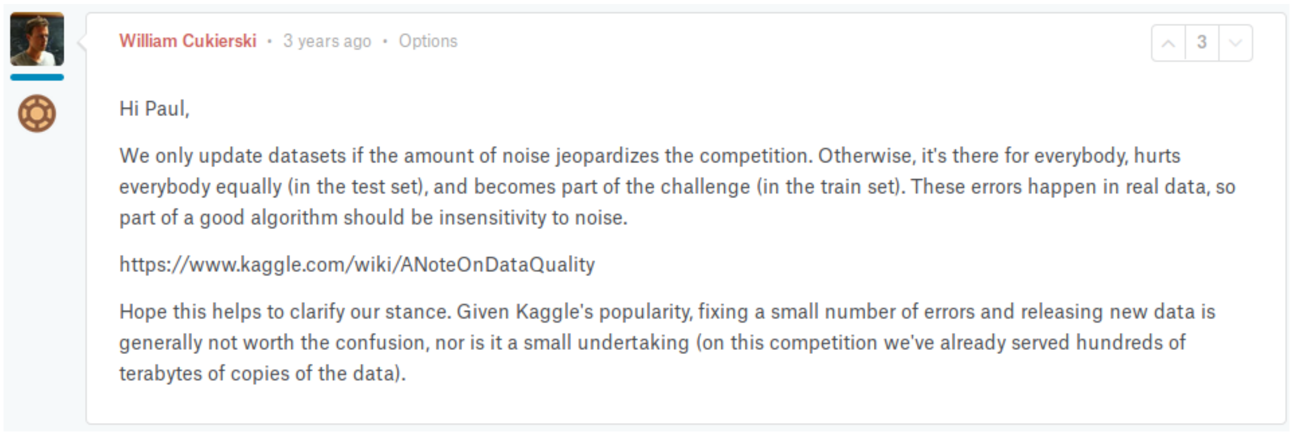
Frequency:

- Class 0: 0.734783
- Class 1: 0.150658
- Class 2: 0.069550
- Class 3: 0.024853
- Class 4: 0.020156

A small sample of photos from the dataset contain images quality distortions stemming from artifacts during image acquisition, transmission, or storage. There’s a discussion on Kaggle(where the dataset was provided) forums about the issue:

https://www.kaggle.com/c/diabetic-retinopathy-detection/discussion/14402

A member of staff clarifies that the the algorithm should be insensitive to noise to a certain degree.



William Cukierski • 3 years ago • Options

Hi Paul,

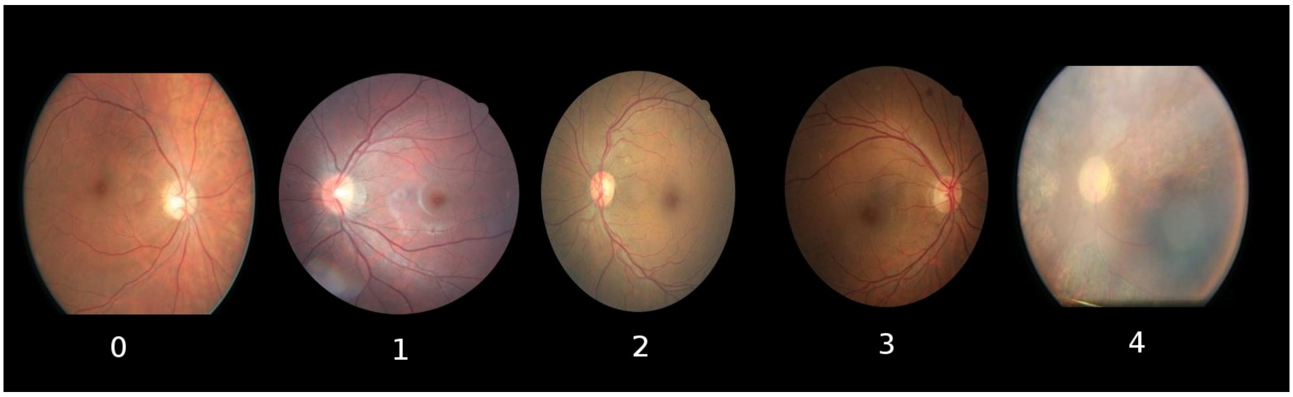
We only update datasets if the amount of noise jeopardizes the competition. Otherwise, it's there for everybody, hurts everybody equally (in the test set), and becomes part of the challenge (in the train set). These errors happen in real data, so part of a good algorithm should be insensitivity to noise.

https://www.kaggle.com/wiki/ANoteOnDataQuality

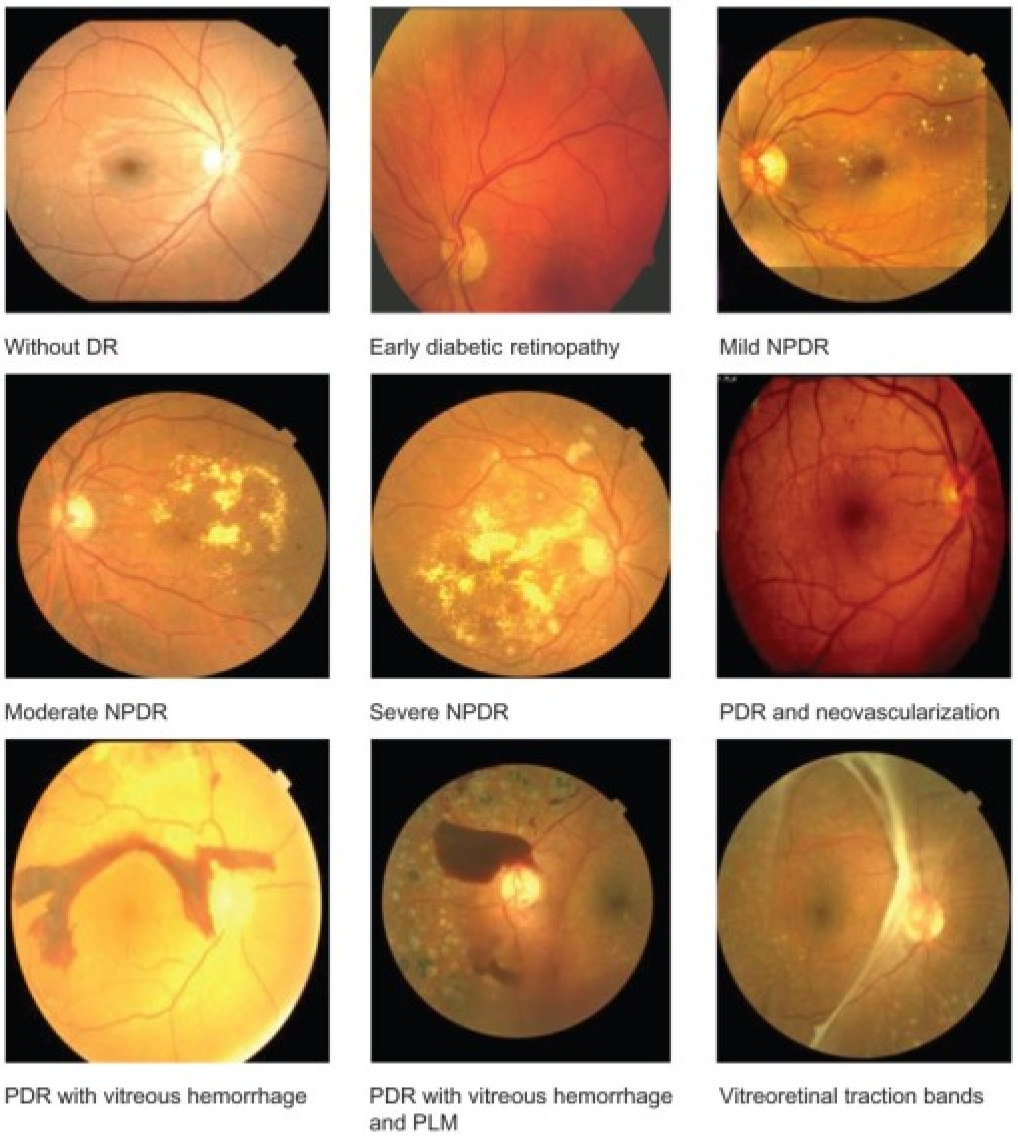
Hope this helps to clarify our stance. Given Kaggle's popularity, fixing a small number of errors and releasing new data is generally not worth the confusion, nor is it a small undertaking (on this competition we've already served hundreds of terabytes of copies of the data).

As the key metric, is the harmonic average of the precision and recall, the small sample of low-quality photos shouldn’t greatly affect the score.

Dataset Examples (*fig.01)*



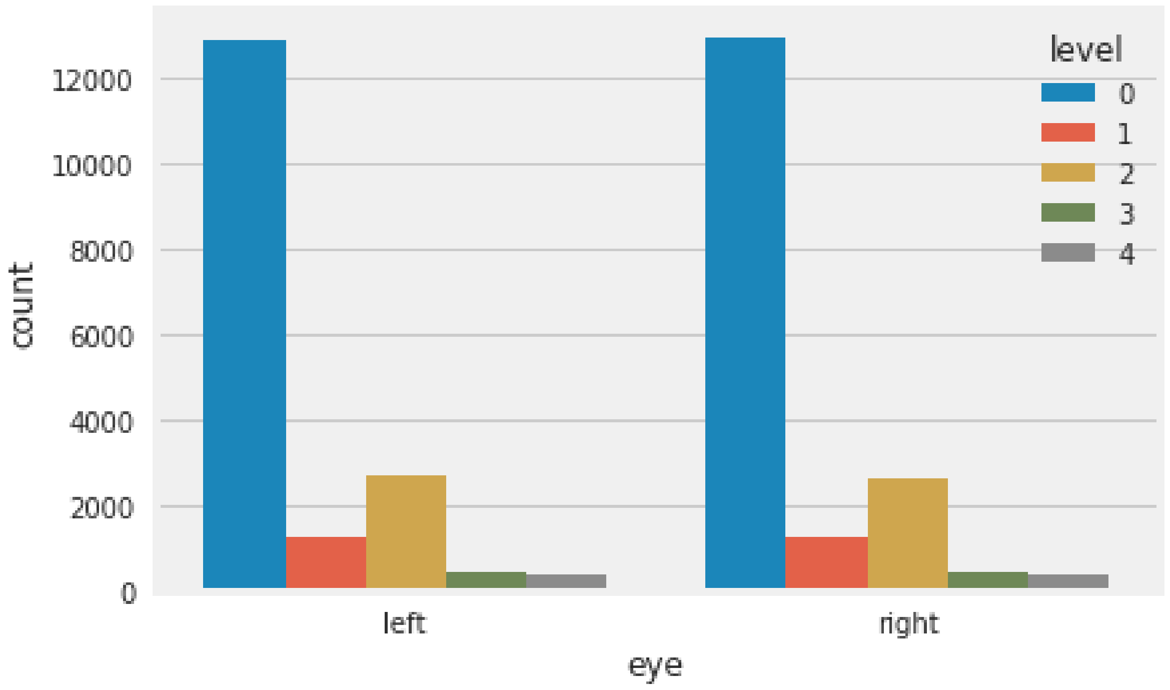
(fig.01)


Without DR
Early diabetic retinopathy
Mild NPDR
Moderate NPDR
Severe NPDR
PDR and neovascularization
PDR with vitreous hemorrhage
PDR with vitreous hemorrhage and PLM
Vitreoretinal traction bands

(fig. 02 : source: https://openi.nlm.nih.gov/detailedresult.php?img=PMC3284208_opth-6-269f3&query=&it=xg&req=4&npos=4)

Dataset specifications:

- **format:** *.jpeg
- **size:** average 3000 x 2000 pixels
- **color channel mean:** (R:95, G:71, B:55)



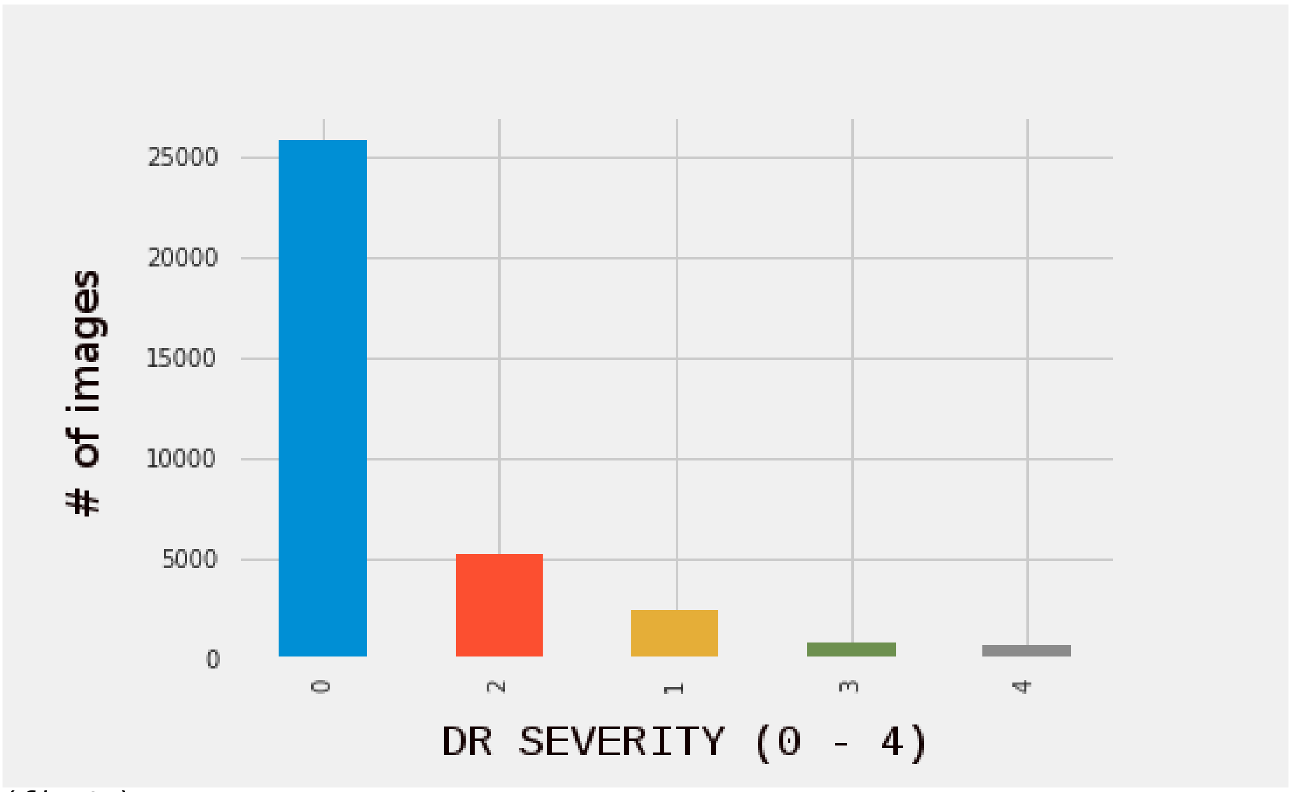
(fig.03)

The ratio of left:right eye images is **1:1.**(see *fig.03)*

Exploratory Visualization

(*please visit Exploratory Data Analysis.ipyn*b)

The image count by class is as follows(see *fig.04)*



(fig.04)

The image ratios will lead to imbalance if the dataset is used without balancing class weights. The majority of images in the training set are classified as 0 (79.5%).

Balancing is done by fitting the model with the following code:

- class_weight={0: 1, 1: 10.6, 2: 4.6, 3: 30, 4: 37}

The mean of colours by channel for sample images are (R:94, G:71, B:55).

Some points to note:

- Individual-level data including age and sex were not available for the dataset,
- Unique patient codes (deidentified) were unavailable for the dataset,

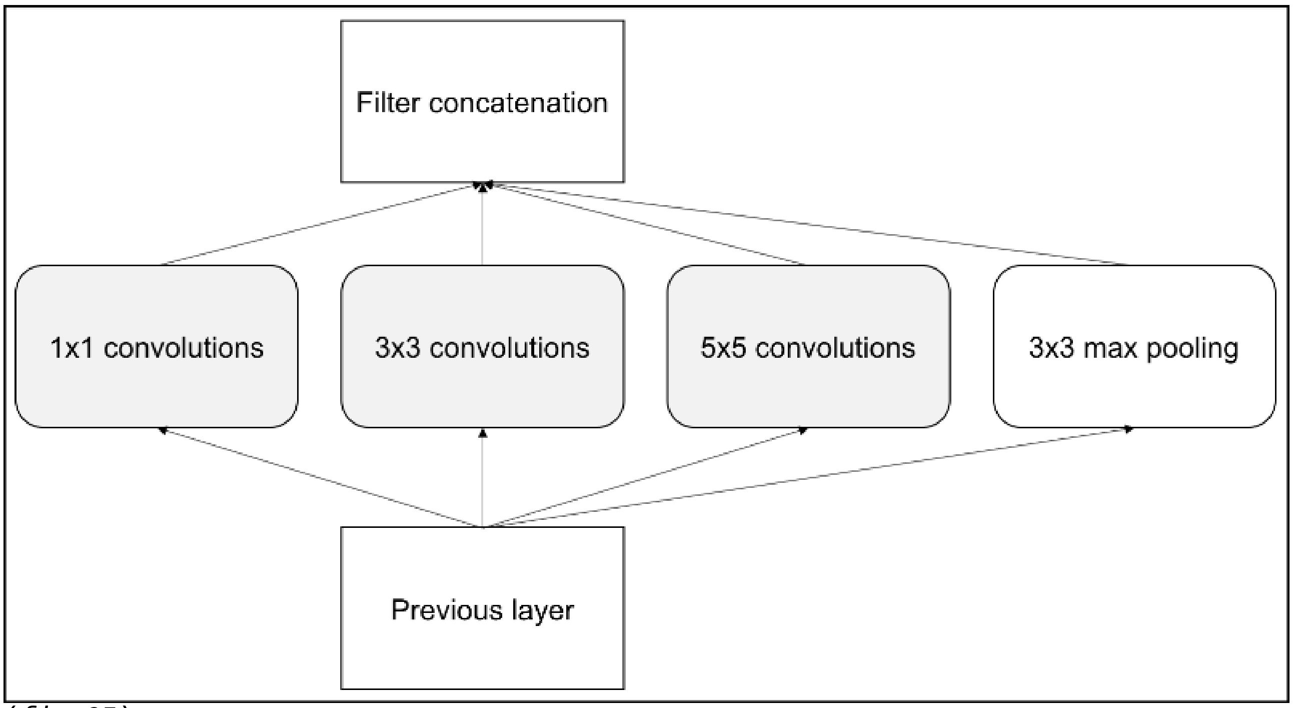
Algorithms and Techniques

This problem is a multiclass problem which will be solved with a convnet, a class of deep, feed-forward artificial neural networks that has successfully been applied to analyzing visual imagery. (source: https://en.wikipedia.org/wiki/Convolutional_neural_network)

Inception-V3 introduced the concept of inception that has a better way of generalization for Convolutional Neural Networks. This was the architecture that won the ImageNet competition in 2014. It is geared towards efficiency for speed and size. It has 12 times lesser parameters than AlexNet. Inception is the micro-architecture on which a macro-architecture is built. Each hidden layer has a higher-level representation of the image. At each layer, we have an option of using pooling or other layers. Instead of using one type of kernel, inception uses several kernels. An average pooling is followed by various size convolutions and then they are concatenated. (*source: Deep Learning for Computer Vision – Rajalingappaa Shanmugamani, Packt Publishing, 2018*)

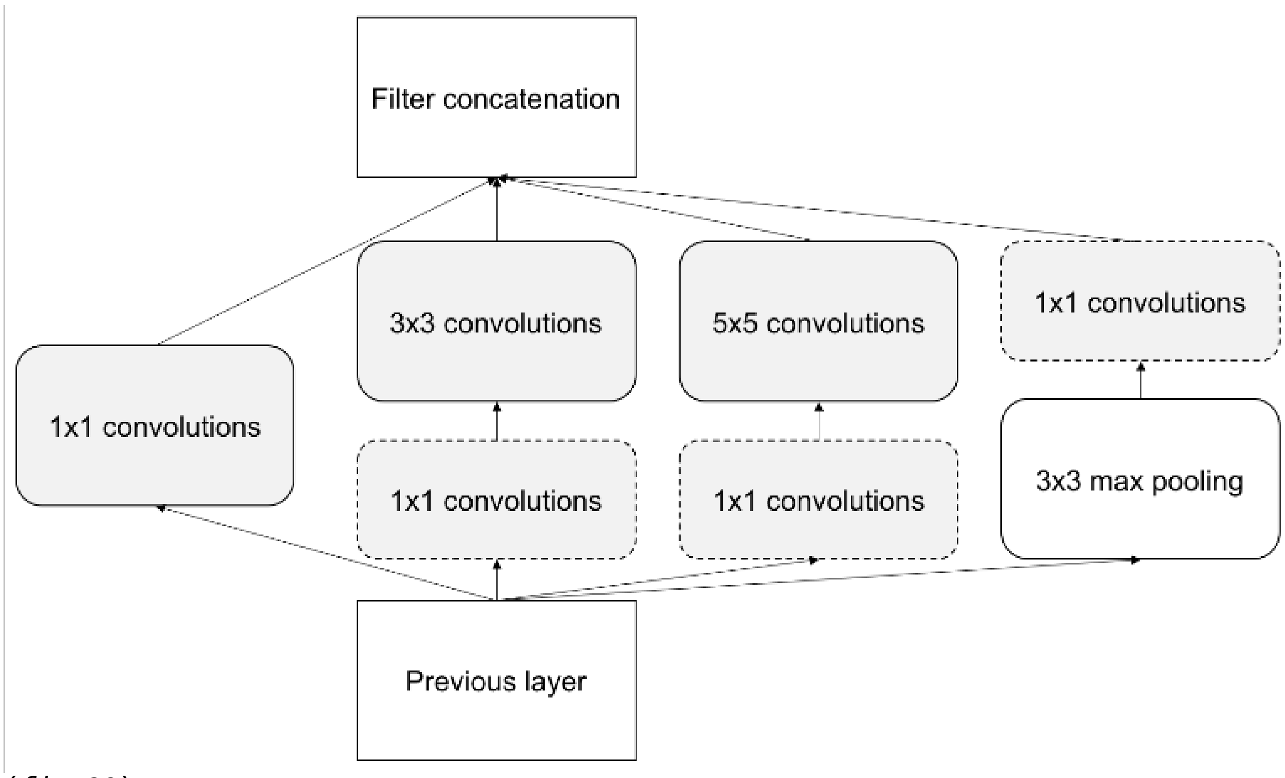
The kernel parameters can be learned based on the data. Using several kernels, the model can detect small features as well as higher abstractions. The 1 x 1 convolution will reduce the feature and, hence, computations. This takes less RAM during inference. The following is the inception module in its simplest form where there are options of convolutions with various kernel sizes and pooling:

(*see fig.05*)



(*fig.05*)

Operations in InceptionV3 happen in parallel, as opposed to AlexNet or VGG. The output volume is huge, and hence, 1 x 1 filters are introduced for dimensionality reduction. When the reduced dimensions are added to the architecture it becomes as follows:



(fig.06)

The network will be trained on 30, 000 training images and validated with 5120 validation images.

Benchmark

The benchmark model will be trained on 30, 000 training images and validated with 5120 validation images. Testing will be done by classifying **1000 testing images**.

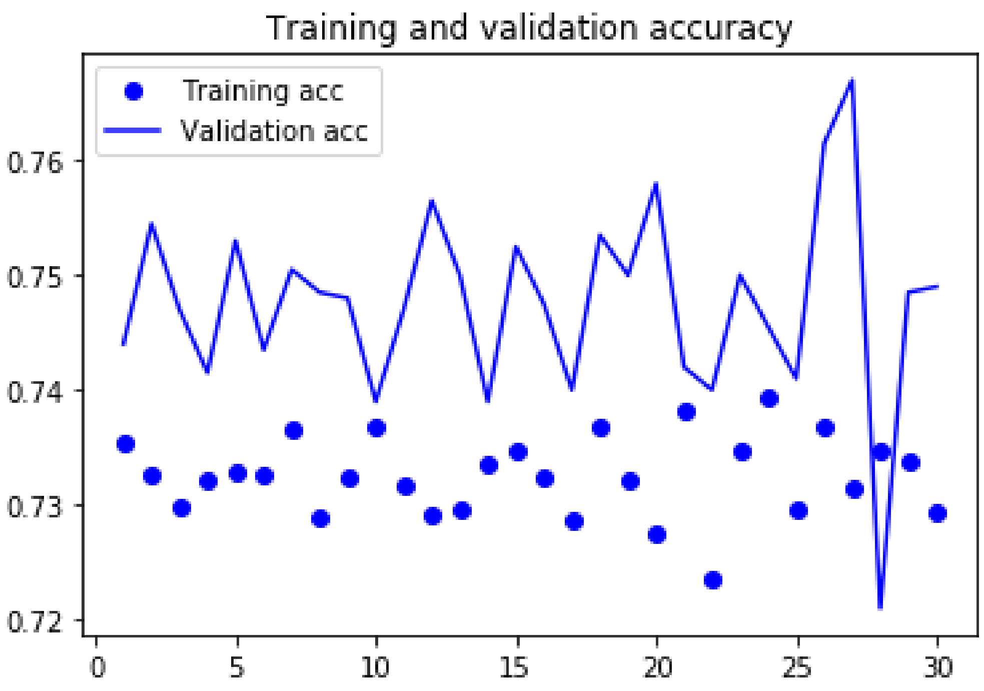
The best scores from the benchmark model is:

TRAINING

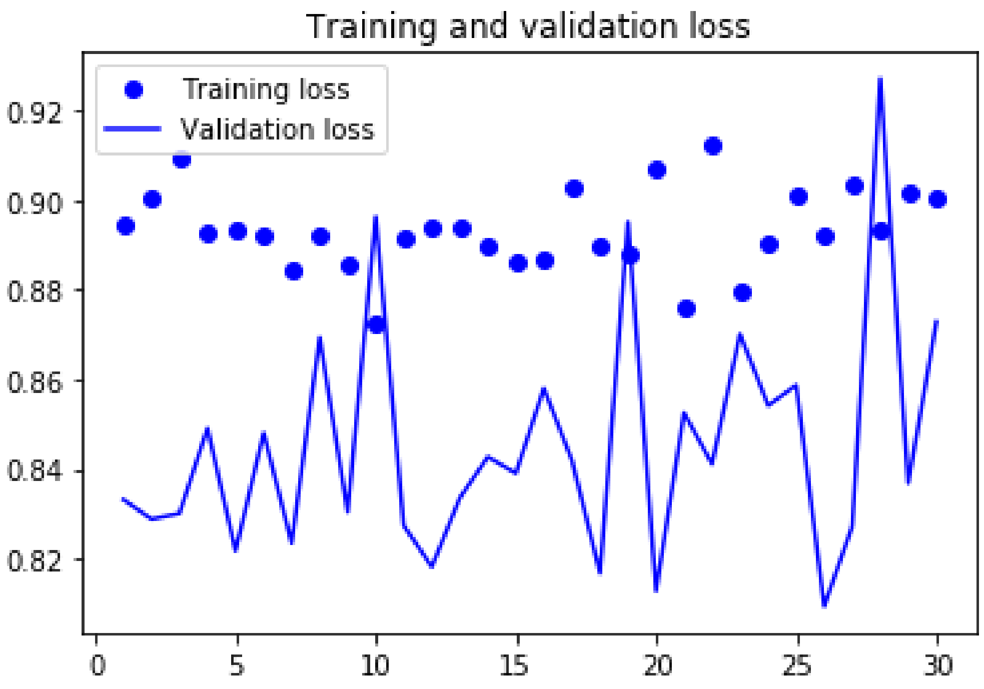
- accuracy: 0.7394
- loss: 0.8724
- **F1: 0.7394**

VALIDATION

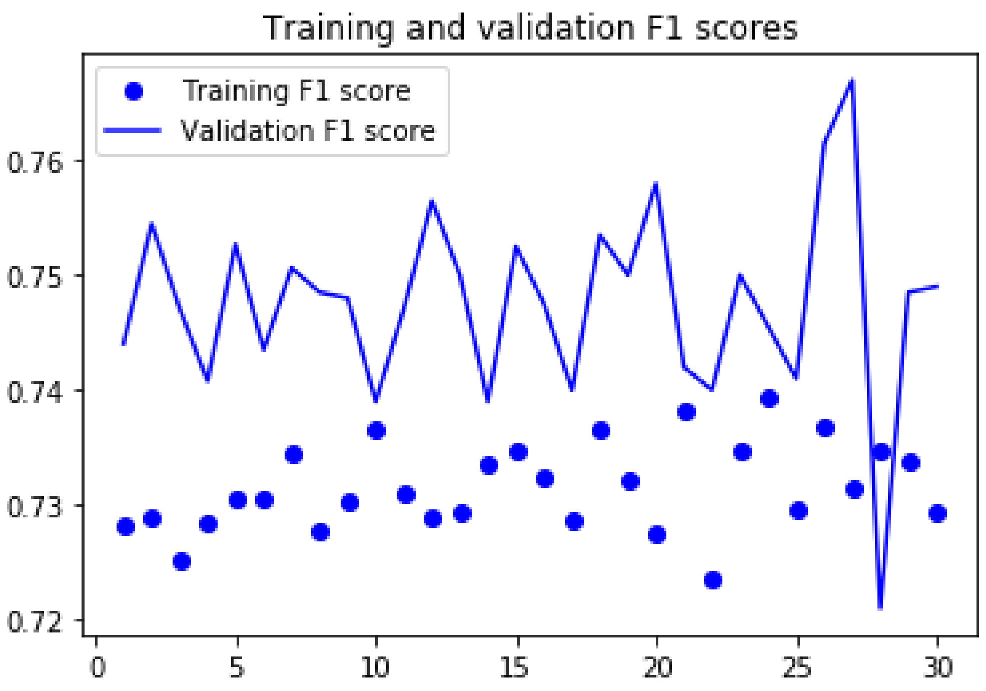
- accuracy: 0.7670
- loss: 0.8094
- **F1: 0.7670**



(fig.07: benchmark model training/validation accuracy)



(fig.08: benchmark model training/validation loss)



(fig.09: benchmark model training/validation F1 scores)

Data Pre-Processing

Images in the dataset:

- vary in size and resolution
- have black borders
- are potentially noisy and with artifacts

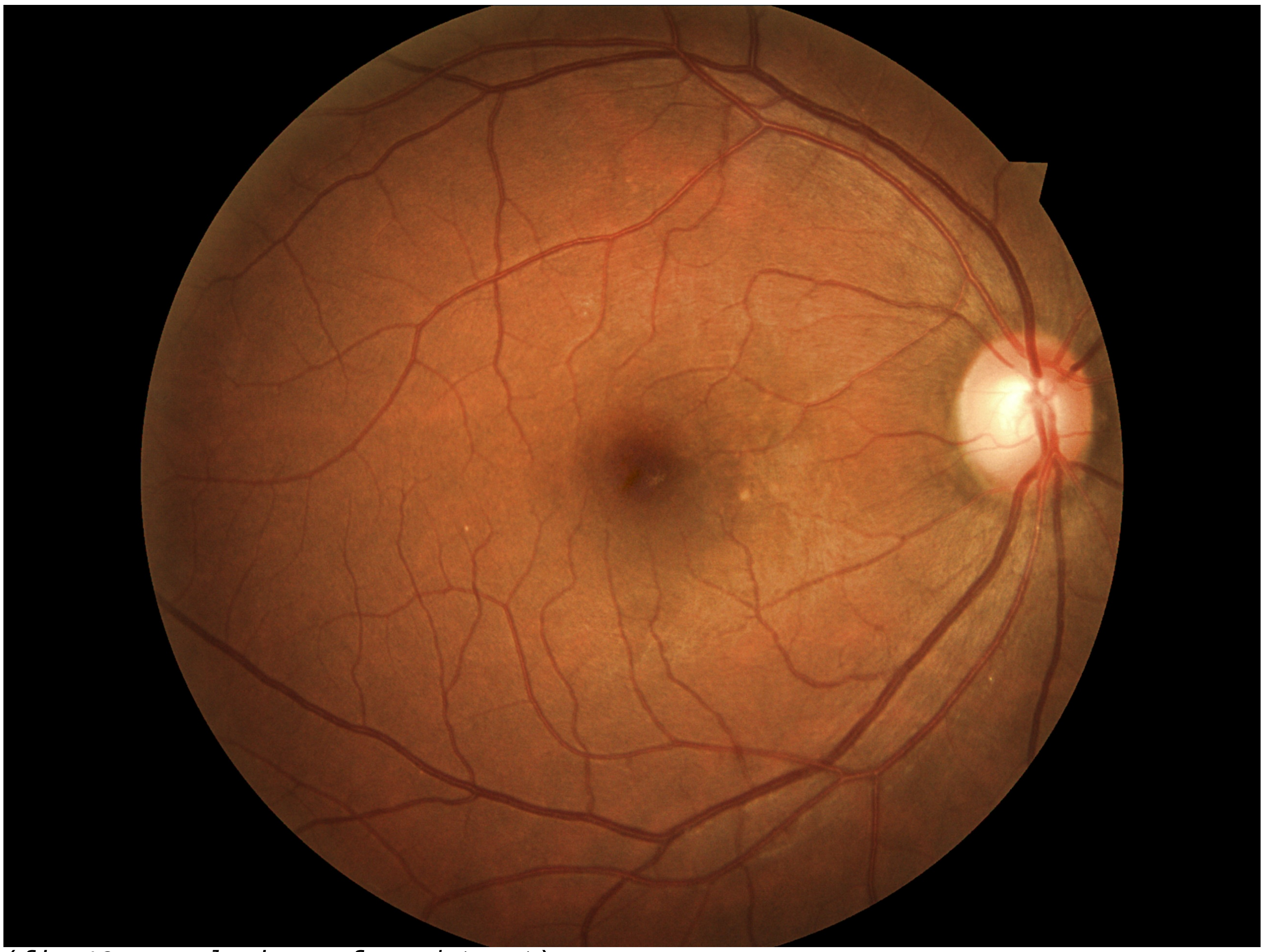

(fig.10: sample image from dataset)

Image processing will be done with a modified script from Tensorflow Model - Research/Slim library:

- https://github.com/tensorflow/models/tree/master/research/slim/datasets

Preprocessing scripts available within the submission folder:

- data_utils.py
- dataset_utils.py
- download_and_convert_data.py
- download_and_convert_diabetic.py

The file **download_and_convert_data.py** will be used to build the DR dataset. The following command cab be run:

```
$ python download_and_convert_data.py --dataset_name diabetic –
dataset_dir /dataset/diabetic
```

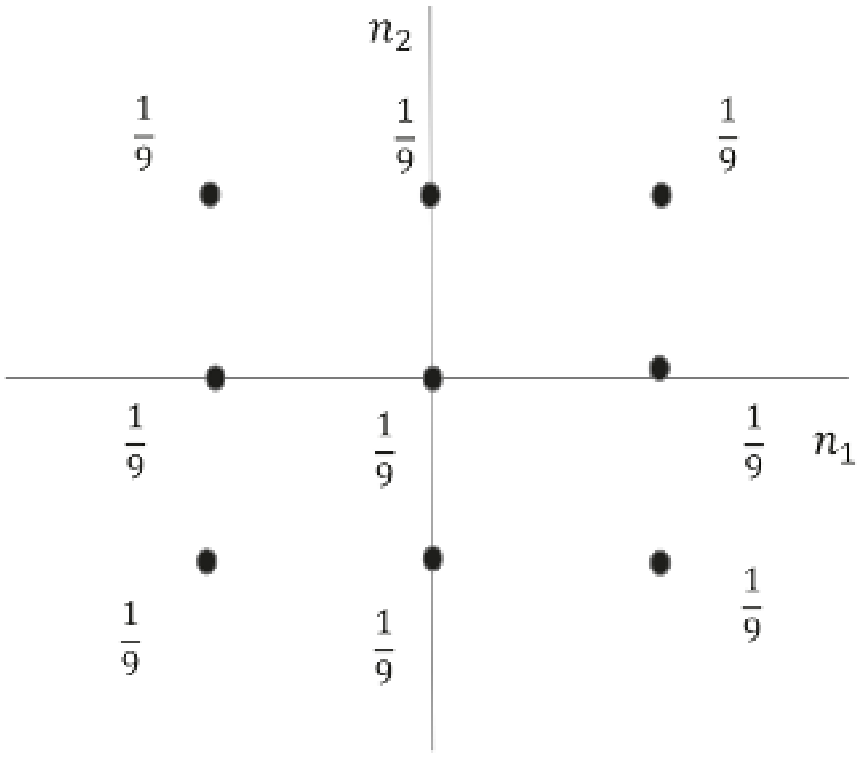
The ‘dataset/diabetic’ folder will contain the ‘trainLabels.csv’ files as well as the 35,120 images downloaded from Kaggle

The **data_utils.py** file will have code responsible for preprocessing the data. The **download_and_convert_diabetic.py** is a conversion of the **download_and_convert_flowers.py** file from the Tensorflow Slim library

Images will undergo transformations in the following order:

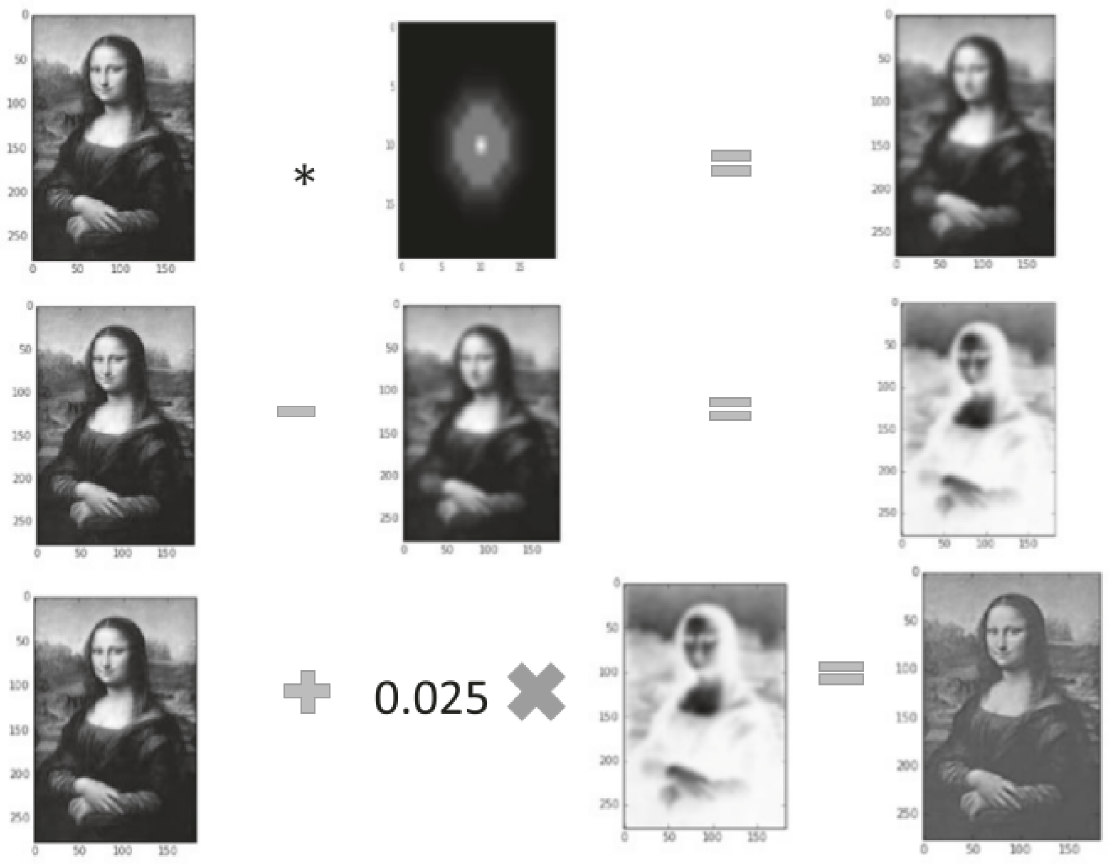
1. denoising (Mean filter) : The Mean filter or Average filter is a low-pass filter that computes the local average of the pixel intensities at any specific point. The impulse response of the Mean filter can be any of the form seen here

$$\begin{bmatrix} 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \end{bmatrix}$$



(*fig.11: Mean Filter, source: Pro Deep Learning with Tensorflow, Santanu Pattanayak, Apress*)

2. sharpening (Gaussian filter)

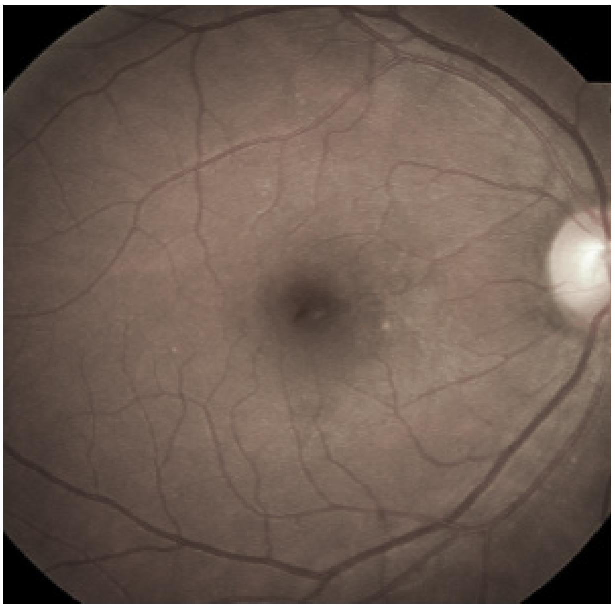


(*fig.12: Gaussian Filter, source: Pro Deep Learning with Tensorflow, Santanu Pattanayak, Apress*)

3. cropping

4. resize to **299 x 299**

Below is an example of a fundus image after pre-processing.



Implementation

Brief explanation of each parameter listed below:

Batch size: Number of samples per gradient update.

Epochs: Number of epochs to train the model. An epoch is an iteration over the entire x and y data provided

Steps per epoch: Total number of steps (batches of samples) before declaring one epoch finished and starting the next epoch

Validation Steps: Total number of steps (batches of samples) to validate before stopping. Only relevant if steps_per_epoch is specified

Benchmark model

The initial solution is a CNN model with the following architecture:

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 148, 148, 128)     3584
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 146, 146, 128)     147584
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 73, 73, 128)       0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 71, 71, 64)        73792
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 35, 35, 64)        0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 33, 33, 64)        36928
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 16, 16, 64)        0
_________________________________________________________________
flatten_1 (Flatten)          (None, 16384)             0
_________________________________________________________________
dropout_1 (Dropout)          (None, 16384)             0
_________________________________________________________________
dense_1 (Dense)              (None, 512)               8389120
_________________________________________________________________
dropout_2 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_2 (Dense)              (None, 256)               131328
_________________________________________________________________
dropout_3 (Dropout)          (None, 256)               0
_________________________________________________________________
dense_3 (Dense)              (None, 128)               32896
_________________________________________________________________
dropout_4 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_4 (Dense)              (None, 5)                 645
=================================================================
Total params: 8,815,877.0
Trainable params: 8,815,877.0
Non-trainable params: 0.0
_________________________________________________________________
```

Parameters/settings for benchmark model:

Batch size: 20

Epochs: 30

Steps per epoch: 500

Validation Steps: 100

Difficulties:

While it’s accuracy score on the validations set is 76%, it classified every image as ***Class 0, No Diabetic Retinopathy.***

The results from the first model will be discounted as it was unable to classify any of the images few into the model.

Final model

The InceptionV3 model has the following architecture:

```
Layer (type)                 Output Shape              Param #

=================================================================
inception_v3 (Model)         (None, 2048)              21802784
_________________________________________________________________
dense_2 (Dense)              (None, 5)                 10245
=================================================================
Total params: 21,813,029.0
Trainable params: 21,778,597.0
Non-trainable params: 34,432.0
_________________________________________________________________
```

Parameters/settings for final model:

Batch size: 20

Epochs: 30

Steps per epoch: 100

Validation Steps: 100

Loss: 'categorical_crossentropy'

Difficulties:

Even with Tensorflow running on a GPU, training the model took a very long time with each epoch.

Model Evaluation and Validation

The InceptionV3 model was chosen after reading the paper published by the Google Research team. The model produced prediction for all of the images in the testing set.

The best scores from the InceptionV3 model is:

TRAINING

- accuracy: 0.4215
- loss: 4.4361
- **F1: 0.4048**

VALIDATION

- accuracy: 0.7661
- loss: 1.1813
- **F1: 0.7634**

However, as this problem's solution is in the domain of medial diagnosis, I wouldn't trust the model without extensive testing and an accuracy of at least 90% and above.

The final result for the network is **77% accuracy** and an **F1 score of 0.76** which isn't robust enough due to the criticality of the network's usage. When the field of practice is medicine, there's no room for error when someone's well-being is on the line.

To get a better accuracy score, a different criterion might be used for evalution, for example, a confusion matrix.

While the large amount of available training data (32K images) means the network is less sensitive to outliers, the dataset should also be balanced and this is discussed in the **Refinement** section.

Refinement

The following parameters were adjusted for the final model:

- class weights for a more accurate representation of the training set,
- image size was increased to (299, 299) from (150, 150),

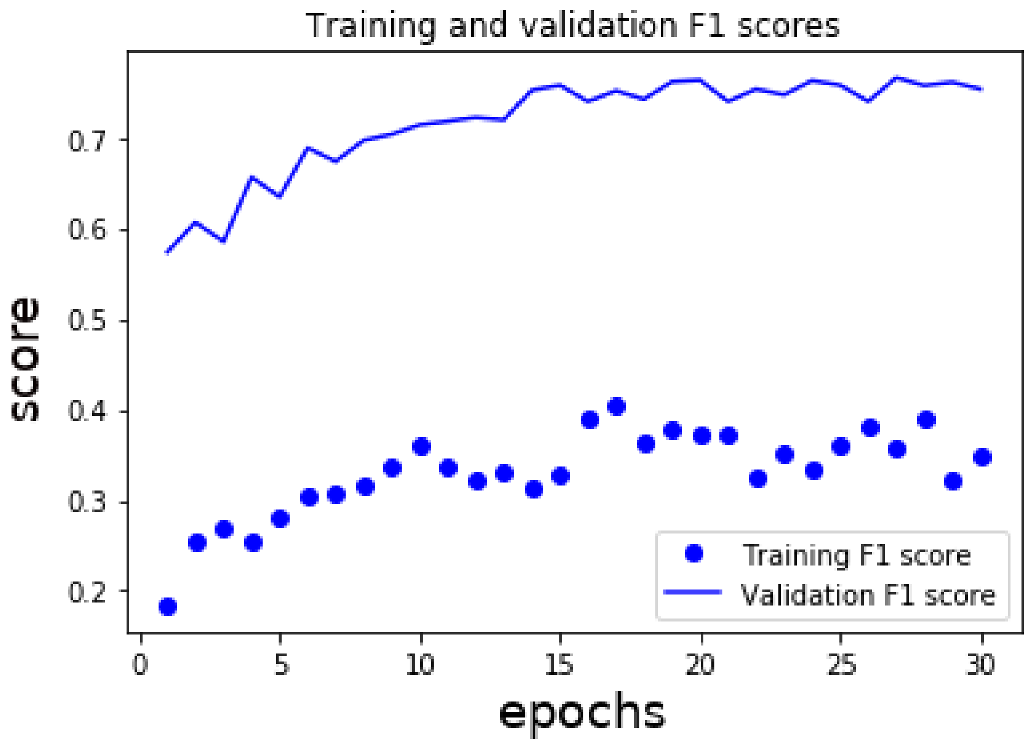
Training images were processed again, with sharpening(Gaussian Filter) and denoising (Mean Filter) applied to the second batch. Processing the second batch took > 8 hours vs. 2 hours for the first batch.

Justification

The F1 scores for the benchmark and final model are almost similar. **The final model is not an improvement over the benchmark model**.

- Benchmark model’s F1 score : **0.7670**
- Final model’s F1 score: **0.7634**

The final model(InceptionV3) had a lower F1 score than the benchmark model and training it for longer periods of time would probably not impact the F1 score based on how the score stabilized after the 15th epoch.



The possible steps to improve the accuracy of the model and it’s F1 score is discussed in the ‘Improvements’ section.

Conclusion and improvements

Free Form Visualization

(please refer to the file free_form_viz.html)

A visualization of how a single conv2d layer with activation and pooling learns from a *processed* image. The processed image's (samples/processed.jpeg) features would give a clue how effective the pre-processing pipeline was.

Reflection

After exploratory data analysis was conducted, the data was preprocessed with scripts from Tensorflow Slim's library. The open-sourced library reduced the amount of time spent on writing code for data cleaning and the preprocessing pipeline significantly. Time wasn't spent building a pipeline from scratch.

Visualizing intermediate activations and benchmarking was performed after exploratory data analysis. When the benchmark model's training was complete, the model classified every image in the testing folder as 0 (Normal/Healthy) even though the images were sourced from classes 1, 2, 3 and 4 as well. Even after class-weight adjustment, the model graded photos seemingly at random. Visualizing activations was a fun aspect of the project. In hindsight, I probably should have visualized activations ***before*** preprocessing.

The first model built from transfer learning was done with VGG16, and VGG19 and ResNet50 subsequently. The VGG16 model had a low F1 score even after training it between 24 -36 hours for 1000 epoch with 1000 validation steps. ResNet was slower to train and produced similar results.

After additional research and uncovering a paper on a similar project(https://jamanetwork.com/journals/jama/fullarticle/2588763) with a Tensorflow backend by the Google Research team, I settled on InceptionV3.

The final F1 score of the InceptionV3 model is slightly lower than the benchmark model's F1 score. It's disappointing to get results which are similar to the benchmark model with a transfer learning model. I would have loved to test the other pre-trained networks if I had more resources; time and access to a powerful GPU.

Improvements

1. Use an optimal image resolution – testing with various image resolutions and using the resolution that produces the best results with InceptionV3.

2. Use all the available image information - the pre-processing stage of this project is key to getting good results. Perhaps certain features in fundus images could be emphasized for easier detection.

3. Use QWK as a loss function - for multi-class classification the standardized loss function to use is the logarithmic loss [19]. In [9] is shown that for ordinal regression problems, where not only a multi-class classification is taking place but also there is possible to establish a sorting of the classes based on some hidden underlying causes, QWK-loss can also be used with better results. (source: https://arxiv.org/pdf/1712.08107.pdf)

4. Use a efficient number of features – the model should be run with better resources so a bigger network can be used.

5. SMOTE (Synthetic Minority Over-Sampling Technique) and balance the dataset by replicating the data for less frequent classes (1, 2, 3 and 4)

6. Write a preprocessing script to remove images with a low mean value RGB score – (e.g. RGB values lower than 50 {R:32, G: 24, B: 50} which means the image is completely dark).