Monday, November 14th, 2022

DIABETIC RETINOPATHY (DR) DETECTION

Application for Ophthalmology

By Pablo Ruiz Lopez

Springboard DSCT Bootcamp

Early an accurate detection: vision saver in diabetic patients

Summary

In this project, we analyzed and processed 1200 retinography images acquired by 3 ophthalmology departments using a color video 3CCD camera mounted on a Topcon NW6 non-mydriatic retinography with 45-degrees field of view. The data was downloaded from the Messidor Database with the main objective of using a deep learning algorithm to detect the level of retinopathy in diabetic patients.

First, we got all data well extracted since it was taken from zip files containing 100 .tif images each and a spreadsheet with the labels for each image. We extracted and saved all images by train and validation datasets in a local Windows path by making use of Python for proper interaction with the operative system. As a first step, we applied a manual data augmentations step that consisted in randomly rotating labeled images (both horizontally and vertically) and applying Gaussian filters to some others. This was made for leveling class balance (see Image 1), so we ended up with **1,309 labeled images**.

After this step we did an extensive image properties' analysis; exploratory, analytical, and tabular data analysis to determine which operations / transformations were relevant for applying to the images and tabular data. The main objective was to prepare the data for inputting it to the model considering computational power and level of accuracy. Afterwards, we did other (fundamental) data processing steps like storing the images in training and validation sets in the form of NumPy arrays (instead of .tif files) to keep the computational processing as light as possible.

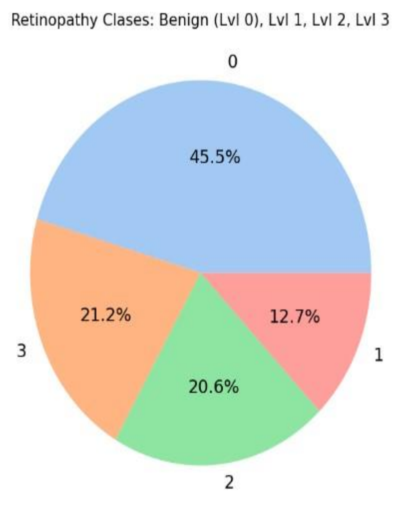
The labels and clinical meaning that were given and selected for the training and modeling stage were

Level 0 (Class 0): Normal (healthy) eye

Level 1 (Class 1): five or less microaneurysms and no internal bleedings found.

Level 2 (Class 2): fifteen or less microaneurysms or no less than five internal bleedings and no neovascularization.

Level 3 (Class 3): Between five to fifteen microaneurysms or more than five internal bleedings and presence of neovascularization.



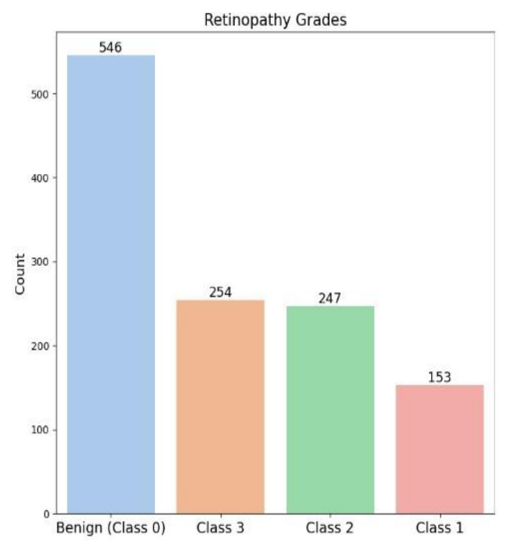


Image 1 – Class balance pie and bar charts

The images given were in RGB-3D format with four different sizes: (1488, 2240, 3), (1435, 2304, 3), (960, 1449, 3) and we decide to leave the third color dimension as it was but reshaping the size by 224x224 for inputting them to them model.

As a benchmark, we tried first to a model using only very basic preprocessing on the images like reshaping and normalization / standardization by mean and standard deviation. This was made since it is known that the models can perform very well without too much image preprocessing. In this stage, we also did some image visualization to see if there were clear patterns between images from different classes, but (without a clinical eye) this step was fairly challenging.

Image Preprocessing

Standardization and normalization

In the exploratory data analysis stage, we count the classes to see how well our train data was balanced, finding out that there were many more class 0 images than the other ones. Which led us to do a manual data augmentation on class 1 to keep them balanced, as shown in Image 1. On this stage, we also explored different operations and transformations that could be applied to the images (ultimately NumPy arrays), some of them shown in Image 2.

However, we did only apply a normalization / standardization by making use of the mean and standard deviation calculated by taking into consideration the size of the image and the third dimension of all images in our database:

$$mu = \frac{sums}{(N)}$$

$$sigma = \sqrt{\frac{sumssquared}{N} - mu^2}$$

After that step, we did some more data transformations by making use of the RandomAffine() function in Pytorch lightning library.

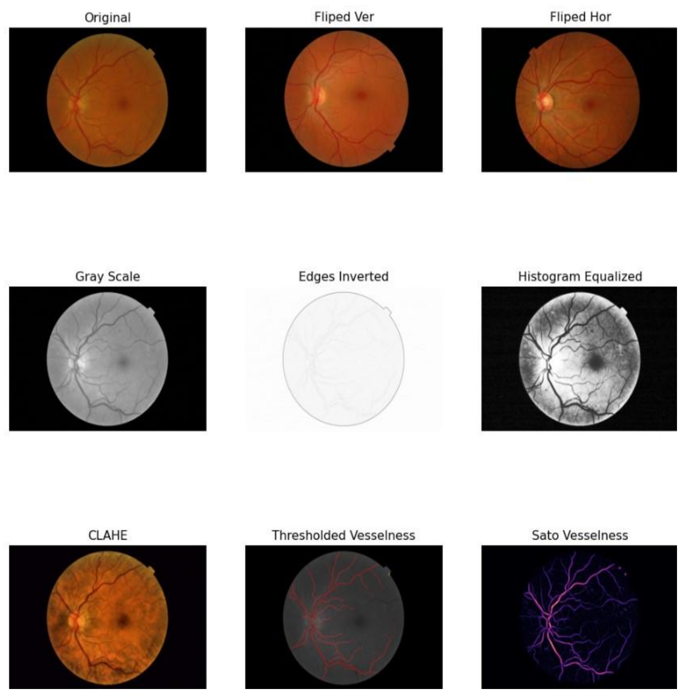


Image 2 – Types of explored operations / transformations made on the images. These include original, flipped vertically, flipped horizontally, gray scaling, edges inversion, histogram equalizing, CLAHE and Vesselness and Sato Vesselness filtering

Afterwards, trainers and loaders were declared as well as computational resources to be used for the model and training, which included the number of batches (10), number of workers (default) and other parameters such as shuffling in the train data.

Label Selection (Retinopathy Grade)

Class imbalance

After manual data augmentation we ended with:

- 546 class 0 images
- 262 class 1 images
- 254 class 2 images
- 247 class 1 images

Preprocessing Made on Images

Normalization / Reshaping

The only preprocessing steps taken were mean **normalization** and std **standardization** on the images and a **224x224x3** reshape on them. These images were saved locally.

Database and Datasets used

Storage

After the EDA, 1309 NPY files were shuffled and separately saved in different train and validation folders which had different classes as subfolders (0, 1, 2, 3). This was made for easy handling during the modeling part.

Model's Key Insights and Results

Analysis of Different Neural Network Architectures

Neural network architectures and model selection

Considering different models for this project

In this final stage of the project, we reviewed different deep learning models to try for our use case. Since this is a computer vision application, we opted to use neural networks to train and build the model. All of this was made using Pytorch library mainly. Above it is shown the summarizes steps we took before we inputted the data to the model:

Data Preprocessing Pipeline
1 Convert image to tensor
2 Normalize image with mean and std
3 Random affine images (only on train data)
4 Convert sets to NumPy arrays
5 Set the loaders with batch sizes of 10 a default number of workers

Table 1: Data Preprocessing Pipeline

This was the initial data preprocessing included at the pipeline. The next step was to load all preprocessed images (1309) to the neural network.

We first tried a very simple Convolutional Neural Network (CNN) for doing a multiclass classification. This neural network consisted of 2 convolutional layers, 2 max pooling layers and 3 fully connected layers. We selected 5 as the number of epochs and trained about 6.7M parameters. Our second model was the same CNN architecture but instead of applying the 3rd transformation in the train data we skipped that part and increased the number of epochs.

Finally, the las model we tried (and selected) was the ResNet18 architecture already built-in and accessible from the library. In the next section we will discuss this architecture in detail. With this network, the model trained 11.5 M parameters. In all our models we used Cross Entropy as our loss function with a learning rate of 0.001 and Adam optimizer

Summary of Model Selection and Performance

Table 2: Neural Networks

Network Architecture	Accuracy	Computation Time (on CPU)
CNN with Random Affine Transformation	42.3%	309 seconds
CNN without RA	57.9%	302 seconds
RessNet18	80.3%	2.5 hours

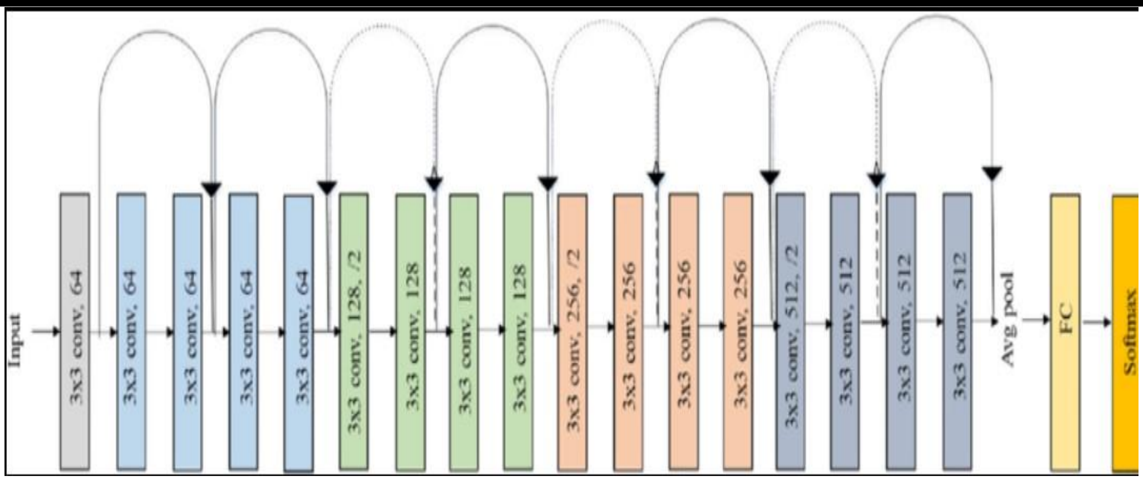


Image 3 – ResNet18 Neural Network Architecture. 17 convolutional layers with different sizes, 1 average pooling layer, one fully connetced layer and one softmax activation at the end. In total there were around 11.5M parameters trained.

Neural Network

Model Results and Insights

ResNet18 Best Performer

As we see in Table 2, our ResNet18 Model (architecture shown in Image 3) was by far the best performer in terms of accuracy. The reason behind selecting accuracy as our metric of evaluation was because the dataset was quite balanced after the manual data augmentation steps.

It is true that the training time was far superior compared to the simple CNN's we tried, but this is due to the network architecture and complexity in the computations (which can eventually be addressed by making use of GPUs on the training stage).

However, there are a couple of key insights we can take from this project.

Key Insights from ResNet18

This architecture is well known to perform well for computer vision applications. In this project we were able to prove the effectiveness of transfer learning since we got a much better accuracy by tweaking the model a bit to fit it with our use case application and image sizes.

Also, we were able to prove that the proper selection of data preprocessing is of paramount importance to reach a good performance on the CNN models we tried.

Summary and Future Work

In this section we would like to point out an interesting phenomenon. These were our metrics' results on the ResNet18:

```
| Precision:   | 0.803 |
| Recall    | 0.803 |
| Accuracy    | 0.803 |
```

As we can see, all metrics have the same value. Which is kind of confusing, however, since this problem is a multiclassification problem and the weighted average on each class is the same, it makes sense we got these results. In summary an **80.3% accuracy in classifying retinopathy grade in ocular images is a fair result by using this type of network.**

Future Work

For a future development, it would be worth to

(1) Try different processing tools and operations in the images before inputting them to the model and check for any improvements.

(2) Gather more data from different world regions and train the model to be more generalizable for different populations.

(3) Train the model using GPU or cloud computing resources to decrease the training time by x10.

(4) Deploy this model and its pipeline into a production environment for clinical purposes and research.

(5) Complementing the project by implementing a risk of macular edema classifier