Capstone Project

Gilles SING-CHEONG

Machine Learning Engineer Nanodegree

June, 2019

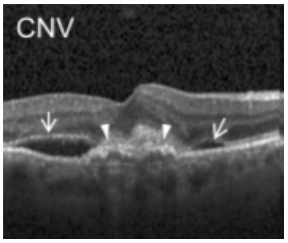
I – Definition

Project Overview

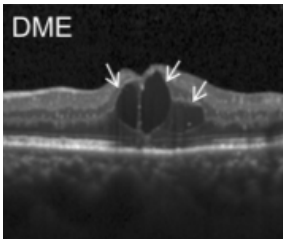
Retinal optical coherence tomography (OCT) is an imaging technique used to capture high-resolution cross sections of the retinas of living patients. OCT has revolutionized the diagnosis and management of common eye diseases such as glaucoma, macular degeneration, and diabetic retinopathy, as well as many other diseases of the retina, optic nerve, and anterior segment of the eye.

Now a day, approximately 30 million OCT scans are performed each year, and the analysis and interpretation of these images takes up a significant amount of time (Swanson and Fujimoto, 2017). The risk of doing a wrong analysis or misenterpretation by an ophtalmologist is present, even if odds should be low. As a support for doctors, automatic diseases classification using Machine Learning (ML) technics are applied to healthcare domain. This is a growing area of research, since few years.

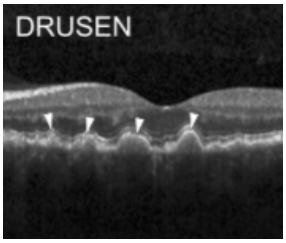
The main goal of this capstone project is to classify automaticaly 3 types of ocular diseases and healthy ocular from OCT images by using ML technics. This capstone will pay attention on the evaluation of a Convolutionnal Neural Network (CNN) model called « SqueezeNet », described below, on the following type of ocular diseases :



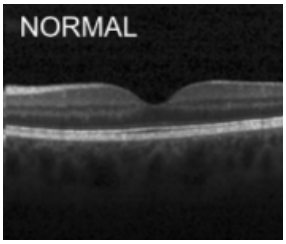
Choroidal neovascularization (CNV) : Choroidal neovascularization is the creation of new blood vessels in the choroid layer of the eye. Choroidal neovascularization is a common cause of neovascular degenerative maculopathy (i.e. 'wet' macular degeneration) commonly exacerbated by extreme myopia, malignant myopic degeneration, or age-related developments.



Diabetic Macular Edema (DME) : DME is a complication of diabetes caused by fluid accumulation in the macula that can affect the fovea. The macula is the central portion in the retina which is in the back of the eye and where vision is the sharpest. Vision loss from DME can progress over a period of months and make it impossible to focus clearly.



Drusen : Drusen are yellow deposits under the retina. Drusen are made up of lipids, a fatty protein. Drusen likely do not cause age-related macular degeneration (AMD). But having drusen increases a person's risk of developing AMD. Drusen are made up of protein and calcium salts and generally appear in both eyes.



Normal : Normal vision occurs when light is focused directly on the retina rather than in front or behind it. A person with normal vision can see objects clearly near and faraway.

Problem Statment

The main objective of this capstone project is to predict from an OCT scan if the patient has retina's disease or not. If yes, which disease is it between CNV, DME, DRUSEN. To achieve an OCT scan classification, many CNN models pretrained or not exist, like VGG16, ResNet and others which are popular for image classification.

For this capstone, we will evaluate « SqueezeNet » model for performance reasons described Algorithms and technics paragraph.

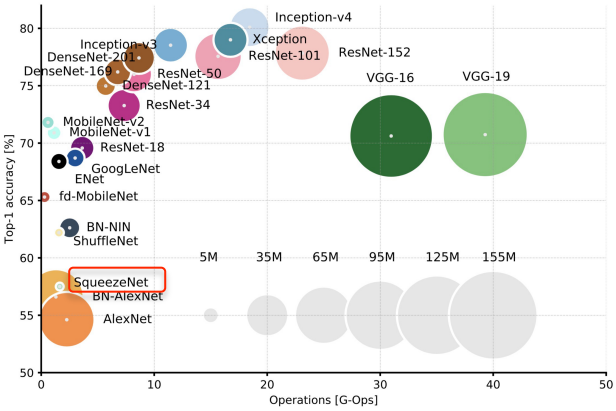


Figure 1 *: Model performance comparison from Medium*

The strategy used to achieve this model evaluation is composed by several steps as :

- Dataset analysis : Analyze number of images in the dataset, which type of image it is.
- Split/Preprocess images : Split and prepare images for the model for training phase.
- Model training : Train the model with prepocessed images for the classification prediction.
- Model evaluation : Evaluate the performance of the model performance with appropriate metrics tools
- Prediction test : Make predicttion with few images.
- Model benchmark comparison : Compare model performance to VGG16 model as reference.

Metrics

As the main goal of this capstone is to identify the presence of a disease based on OCT images, and which one is it, th problem can be assimilated as a multi-classification problem. To evaluate the classification performance, which means how accurate predictions are, several metrics tools are used and described below :

Accuracy

Source : https://scikit-learn.org/stable/modules/model_evaluation.html#accuracy-score

« The accuracy score function computes the accuracy, either the fraction (default) or the count (normalize=False) of correct predictions. » from scikit-learn. »

$$\texttt{accuracy}(y, \hat{y}) = \frac{1}{n_{\text{samples}}} \sum_{i=0}^{n_{\text{samples}}-1} 1(\hat{y}_i = y_i)$$

F1 score

Source : https://scikit-learn.org/stable/modules/generated/sklearn.metrics.f1_score.html#sklearn.metrics.f1_score

f1-score is the harmonic mean of precision and recall. f1-score is comprised between 0 and 1, where 1 is the perfect precision and recall.

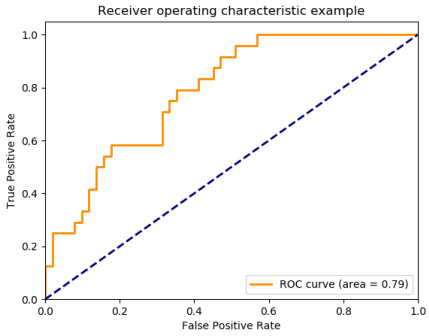
$$f_1\text{-score} = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}$$

« **Precision** is the ability of the classifier not to label as positive a sample that is negative. »
« **Recall** is the ability of the classifier to find all the positive samples. » from scikit-learn.

ROC - Receiver Operating Characteristic

Source : https://scikit-learn.org/stable/modules/model_evaluation.html#receiver-operating-characteristic-roc

A Receiver Operating Characteristic (**ROC**) curveis a graphical plot which illustrates the performance of a model for. It shows the ratio of true positives out of the positives (**TPR** = True Positive Rate) vs. the fraction of false positives out of the negatives (**FPR** = False Positive Rate), at various threshold settings. The area under the curve indicates how good the model is at separating data classification. An area of 1. is considered as a perfect classification.



Sample of ROC Cruve
Picture from sklearn web page

Confusion Matrix

Source : https://medium.com/hugo-ferreiras-blog/confusion-matrix-and-other-metrics-in-machine-learning-894688cb1c0a
https://en.wikipedia.org/wiki/Confusion_matrix

A confusion matrix is a specific table layout showing the performance of an algorithm. Each row of the matrix represents the instances in a predicted class, while each column represents the instances in an actual class.

Confusion Matrix	Predicted: Class A	Predicted: Class B
Actual: Class A	False Positives (FP)	True Negatives (TN)
Actual: Classe B	True Positives (TP)	False Negatives (FN)

- True positives (TP): the cases for which the classifier predicted Class A and the prediction were actually Class A.
- True negatives (TN): the cases for which the classifier predicted Class B but the prediction were actually Class B.
- False positives (FP): the cases for which the classifier predicted Class A but the prediction were actually Class A.
- False negatives (FN): the cases for which the classifier predicted Class B and the prediction were actually Class B.

II – Analysis

Data Exploration

Source : https://www.kaggle.com/paultimothymooney/kermany2018

For this project, the dataset used are issued from Kaggle. It is 84,495 X-Ray images (JPEG) classified in 4 categories (NORMAL,CNV,DME,DRUSEN). Images are labeled as (disease)-(randomized patient ID)-(image number by this patient). For sample : CNV-53018-2.jpeg.

These OCT scans are splitted in 3 parts. One for the model training phase, one for the validation phase and the last one for the testing phase as shown in **Figure 2**.

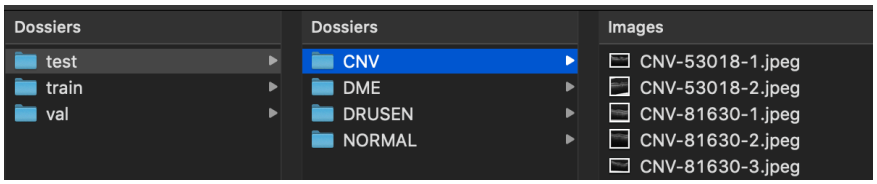


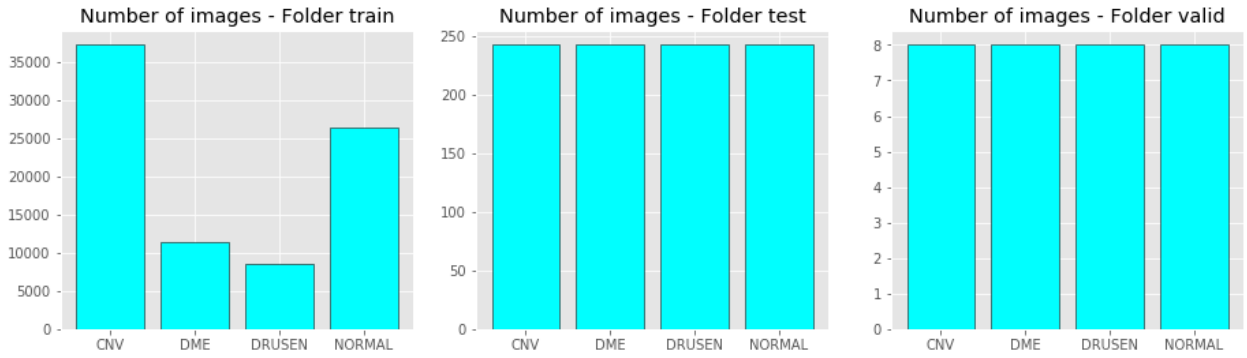
***Figure 2** : Sample of dataset contents*

Exploration Visualization

The 84,495 images are splitted in 3 directories :

- Training dataset : 83484 images
- Testing dataset : 968 images
- Validation dataset : 32 images

Each directory contains images for each ocular diseases and ocular healthy, as described in **Graphic 1**.



***Graphic 1** : Number of images per diseases per dataset folder*

Algorithms and Techniques

Sources : https://arxiv.org/abs/1602.07360
https://medium.com/@subham.tiwari186/squeeze-net-things-you-need-to-know-a1a39c7a75c7
https://github.com/DT42/squeezenet_demo

Nowadays, multiple Deep CNN architectures are focused primarily on increasing accuracy level. For an equivalent precision, a smaller architecture would presents multiple advantages :

- More efficient distributed training.
- Less overhead when exporting new models to clients.
- Feasible FPGA and embedded deployment.

One of models architutecture presenting those advantages is called « SqueezeNet ».

SQUEEZENET MODEL

The goal of Squeeze-net model is to get a similar accuracy level with fewer parameter than other models (**Figure 2**). 3 architectures of Squeeze-net exist :SqueezeNet, SqueezeNet with simple bypass and SqueezeNet with complex bypass (**Figure 3**).

For our capstone, SqueezeNet and SqueezeNet with simple bypass are used. The first one is for model testing as starting point, and the second one is for improvement.

The SqueezeNet model architecture (**Figure 3**) is composed by a convolution layer (‘conv1’) followed by 8 « Fire Modules » (‘fire2-9’), ending with a convolution layer (‘conv10’). The number of filters per fire module increase from the beginning to the end of the network. « Fire Module » (described below) brings down the number of parameters. A Max-pooling layer is used with a stride of 2 after layers conv1, fire4, fire8, and conv10.

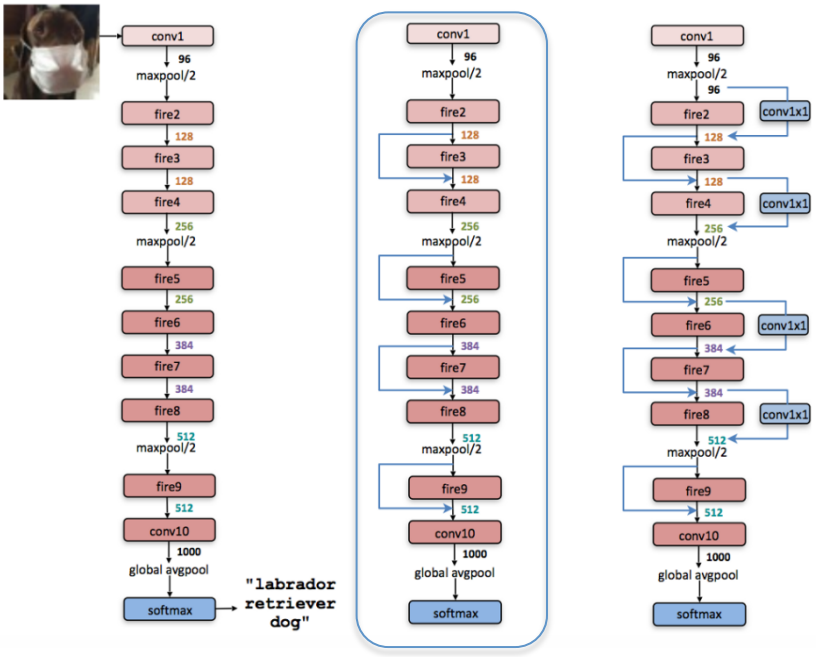


***Figure 3** : Macroarchitectural view of our SqueezeNet architecture. Left: SqueezeNet; Middle: SqueezeNet with simple bypass, Right : SqueezeNet with complex bypass. Image from link*

The SqueezeNet with simple Bypass model architecture is the same as SqueezeNet, but has Bypass connections around Fire modules 3, 5, 7, and 9. These modules are requiered to learn a residual function between input and output.

FIRE MODULE

Fire module, that composed SqueezeNet architecture, brings down number of parameter.

It is composed by a squeeze convolution layer (which has only 1x1 filters) with Relu activation function. It feeds into an expand layer composed by a mix of 1x1 and 3x3 filters convolution with Relu activation function (**Figure 4**).

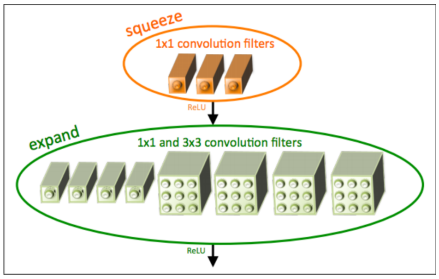


***Figure 4** : Fire Module. Image from link*

ADDITIONAL – KERNEL INITIALIZER

Source : https://arxiv.org/abs/1502.01852

By default, kernel initializer used in keras is « glorot_uniform ». For the project, the kernel initializer used is « he_normal » for convolution layer in « Fire Module », 'conv1' and 'conv10'.

« He normal draws samples from a truncated normal distribution centered on 0 with stddev = sqrt(2 / fan_in) where fan_in is the number of input units in the weight tensor. »

LOSS FUNCTION

Source : https://gombru.github.io/2018/05/23/cross_entropy_loss/
https://peltarion.com/knowledge-center/documentation/modeling-view/build-an-ai-model/loss-functions/categorical-crossentropy

As the goal of the capstone is to identify if an OCT presents diseases and which one it is, we can consider the problem as a multi class classification, and « **Categorical Cross-Entropy** » Loss function is suited for our purpose.

Categorical Cross-Entropy is a combinaison of SoftMax activation function (f(s)) and Cross-Entropy loss function (CE) (**Figure 5**). f(s) turns CNN score into class probabilities, while CE will compare the distribution (the activations in the output layer, one for each class) with the true distribution, where the probability of the true class is set to 1 and 0 for the other classes. The true class is represented as a one-hot encoded vector.

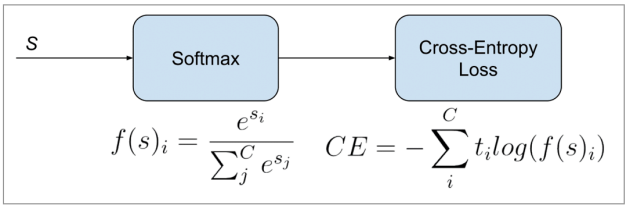


***Figure 5** : Categorical Cross-Entropy*
Image from link

With C for classes, S for score , infered by CNN for each classes, ti for the vector target label.

Benchmark

Some studies comparing models performance had been done with some benchmark in the following link.

The SqueezeNet model will be compared to transfert learning model using pre-trained MobileNet, VGG16 giving 71.2% accuracy after 8 epochs, 87.2% accuracy after 9 epochs and for a custom CNN 5 hidden layers gives 94.4% accuracy after 7 epochs.

Remark : A model using VGG16 was created for testing for model size quantification.

III – Methodology

Data Preprocessing

Source : https://keras.io/preprocessing/image/

The model takes as input images which are preprocessed, called also augmented (**Figure 6**), as described below :

For training images :

- Resizing : original image → 350X350
- Convert image to tensor format for Keras
- Feature wise center and standart normalization : Standardize pixel value across the dataset. Featurewise_center =True, Featurewise_std_normalization=True
- Rotating : Random rotation of the image in the range 20 deg. rotation_range=20
- Width shifting : Random width shifting in the range 0.2. width_shift_range=0.2
- Height shifting : Random Height shifting in the range 0.2. height_shift_range=0.2
- Shearing : Random shear in the range 0.2. shear_range=0.2
- Zooming : Random zooming in the range o.2. zoom_range=0.2
- Horizintal flipping : Random horizontal flipping. horizontal_flip=True
- Filling mode : fill_mode='nearest'

For testing images :

- Resizing : original image → 350X350
- Convert image to tensor format for Keras
- Feature wise center and standart normalization : Standardize pixel value across the dataset. Featurewise_center =True, Featurewise_std_normalization=True

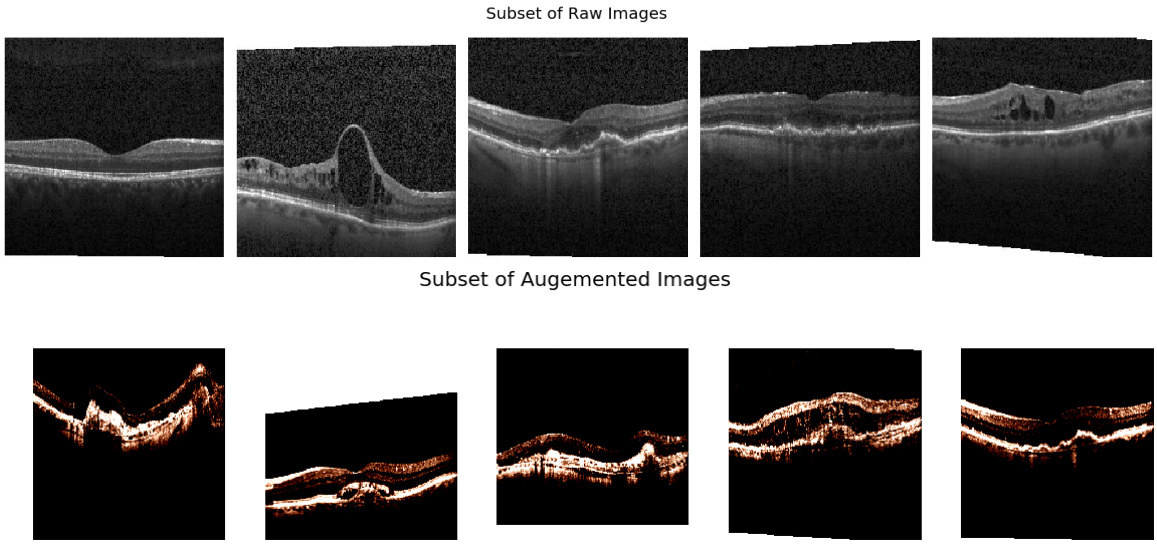
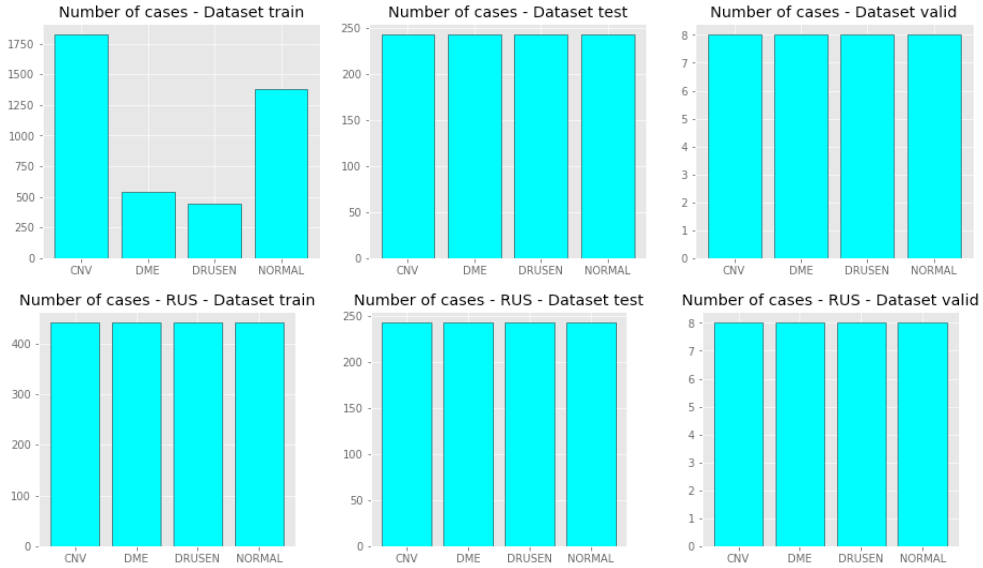


***Figure 6** : Top-Raw images, Bottom-Preprocessed (or Augmented) images*

Implementation

The OCT diseases detection implementation can be break down in several steps :

1. Get files path from directories. Function used : **load_dataset()**.
2. Split training data : Take a few percentage of training data. Function used : **train_test_split ()**
3. RUS random under split : Due to the non equal number of images per diseases in training data, compared to the equal number of images for test and validation dataset, we split the training dataset (**Graphic 2**). Also the split was necessary due to the computing limitation. Function used : **ReSamplingData()**.



***Graphic2** : Top-Images disease distribution before RUS, Bottom-Images disease distribution after RUS*

4. Load and Preprocess images : Load and convert image to tensor format and image class to a one hot coding vector. Function used : **paths_to_tensor(), onehotcoding()**.
5. Create and configure augemented image generator for training and testing dataset.
6. Build SqueezeNet Model with simple bypass (described in Algorithms and technics paragraph), with optimizer, loss function setup and compile model. Function used : **SqueezeNet_InitRandomWeight_ByPass_Model(),fire_module()**.
7. Specify parameters for the training : loss function, optimizer, number of epochs, etc
8. Train model with preprocessed augmented images from training dataset and testing dataset.
9. Save and plot Acc/Loss, Val_acc/Val_loss Vs epoch data. Function used : **HistoryTraining_Save(),plot_model_history()**.
10. Reload the best model.
11. Model evaluation (Classification report, Confusion Matrix, ROC Curve) with testing dataset. Function used : **Classification_ROC_Report(), confusion_matrix(), ROC()**.
12. Prediction for validation with validation data.

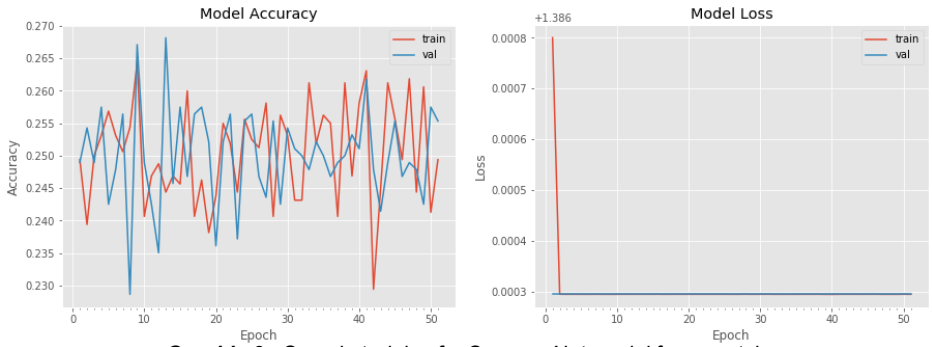
This implementation was done in 2 phases :

- Code implementation and test with few dataset were done on a laptop without GPU.
- The training of models tested for sensitivity studies and for the final model were done on AWS EC2 p2.xlarge machine.

Main code for the final training in Jupyter notebook:
OCT_SqueezeNet_From_Scratch_InitRandomWeight_ImageResize_AugmFeatureWise_MoreData.ipynb

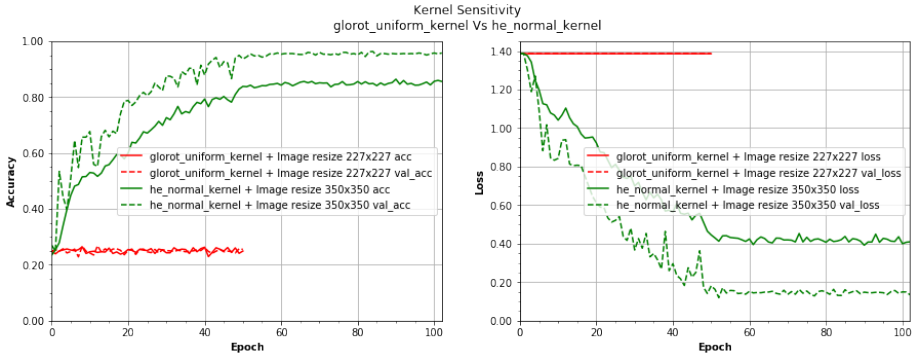
Refinement

To achieve the final results, many iteration were done. Chronologically, the initial model tested was a classical SqueezeNet with defaut keras parameters. The model didn't perform. Loss constant during the training as in **Graphic 3**. Similar results were observed with a pretrained SqueezeNet model. None of models tested learned something.

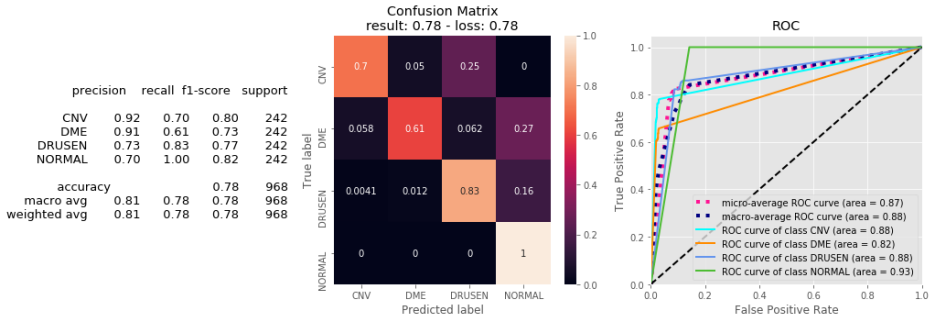


***Graphic 3** : Sample training for SqueezeNet model from scatch*

After investigation, one of the main parameter that permits to have a descency starting point, was the kernel initialization choice, using « he_normal » kernel initialization, combined with the image sizing 350x350. However, even if the model training seems encouraging (**Graphic 4**), the perfomance for prediction is not good enought (**Graphic 5**).



***Graphic 4** : Kernel initializer – glorot_uniform (default in keras) Vs he_normal*



***Graphic 5** : Kernel initializer – glorot_uniform (default in keras) Vs he_normal*

Based on the starting point, sensitivity studies or refinement were done in order to improve the performance of the model. Not all sensitivities tests resutls are presented. Only those presenting significant improvement are shown below:

1. Augmentation featurewise_center and featurewise_std_normalization

Instead of rescaling images during augmentation (**Figure 7**), standardize pixel among dataset (**Figure 8**) technic is used. During the training phase, pixel standardization shows quicker accuracy and loss evolution than rescaling image method (**Graphic 6**). Also it shows a light improvement for classification (**Graphic 7**) compared to the rescaling method (**Graphic 5**).

Jupiter Notebook :

- Image rescaling 1/255 : **OCT_SqueezeNet_From_Scratch_InitRandomWeight_ImageResize.ipynb**
- Image featurewise :
  OCT_SqueezeNet_From_Scratch_InitRandomWeight_ImageResize_AugmFeatureWise.ipynb

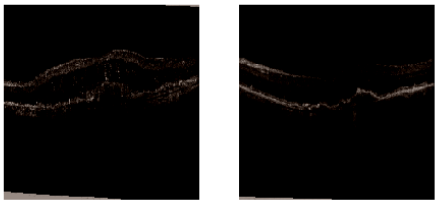


***Figure 7** : Sample image rescaling 1/255*

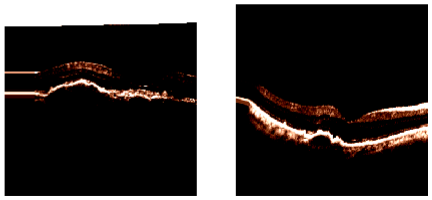
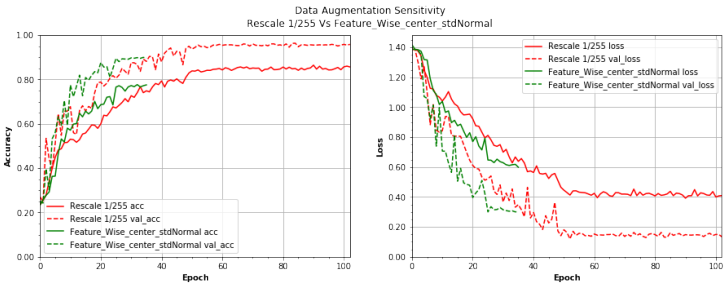
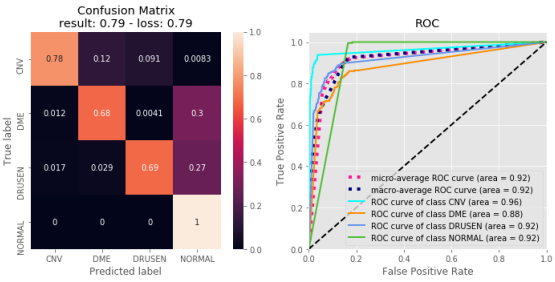


***Figure 8** : Sample image featurewise_center and featurewise_std_normalization*



***Graphic 6** : Data Augmentation – Rescaling Vs Feature wise center, standart normalization*

	precision	recall	f1-score	support
CNV	0.96	0.78	0.86	242
DME	0.82	0.68	0.74	242
DRUSEN	0.88	0.69	0.77	242
NORMAL	0.63	1.00	0.78	242
accuracy			0.79	968
macro avg	0.82	0.79	0.79	968
weighted avg	0.82	0.79	0.79	968



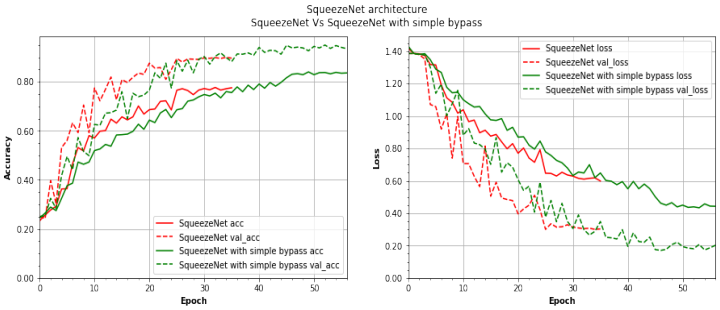
***Graphic 7** : SqueezeNet model evaluatoin with feature wise cener/std normalization*

2. SqueezeNet Simple ByPass Model

From the classical SqueezeNet model, there is another way to improve the performance of the model for the classification by using Bypass (See Algorithms and Techniques paragraph). This approach was described in the SqueezeNet paper. The model with simple Bypass takes more time to inscrease its accuracy and reduce its loss during the training phase (**Graphic 8**). In term of classification performance, the mode with Bypass has improved (**Graphic 9**) compared to the classical SqueezeNet (**Graphic 7**)

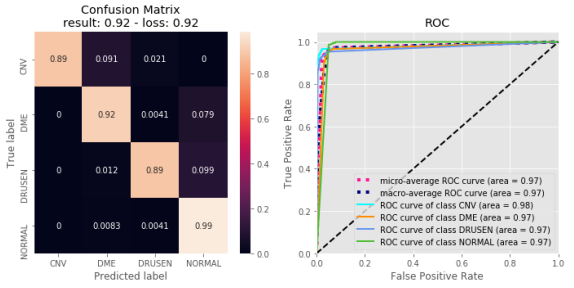
Jupiter Notebook :

- SqueezeNet with simple Bypass:
  OCT_SqueezeNet_From_Scratch_InitRandomWeight_ImageResize_AugmFeatureWise_ByPass.ipynb
- SqueezeNet :
  OCT_SqueezeNet_From_Scratch_InitRandomWeight_ImageResize_AugmFeatureWise.ipynb



***Graphic 8** : Model artchitecture – SqueezeNet Vs SqueezeNet with simple bypass*

	precision	recall	f1-score	support
CNV	1.00	0.89	0.94	242
DME	0.89	0.92	0.90	242
DRUSEN	0.97	0.89	0.93	242
NORMAL	0.85	0.99	0.91	242
accuracy			0.92	968
macro avg	0.93	0.92	0.92	968
weighted avg	0.93	0.92	0.92	968



***Graphic 9** : SqueezeNet with simple bypass model evaluation*

3. Number of images for training dataset : 1636 images, 5008 images

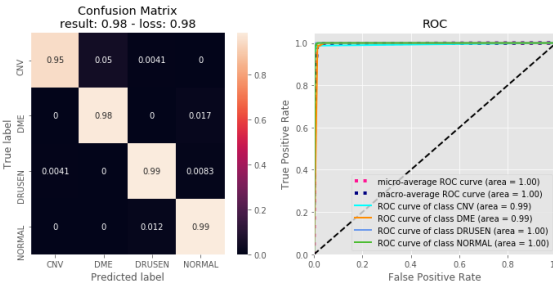
Increasing the number of dataset used for the training permits to improve the training performance (**Graphic 10**), and the classification performance (**Graphic 11**).

Jupiter Notebook :

- 5008 images:
  OCT_SqueezeNet_From_Scratch_InitRandomWeight_ImageResize_AugmFeatureWise_MoreData_ByPass.ipynb
- 1636 images :
  OCT_SqueezeNet_From_Scratch_InitRandomWeight_ImageResize_AugmFeatureWise_ByPass.ipynb

***Graphic 10** : Training dataset size – 5008 images Vs 1636 images*

	precision	recall	f1-score	support
CNV	1.00	0.95	0.97	242
DME	0.95	0.98	0.97	242
DRUSEN	0.98	0.99	0.99	242
NORMAL	0.98	0.99	0.98	242
accuracy			0.98	968
macro avg	0.98	0.98	0.98	968
weighted avg	0.98	0.98	0.98	968



***Graphic 11** : Training dataset 5008 images - SqueezeNet with simple bypass model evaluation*

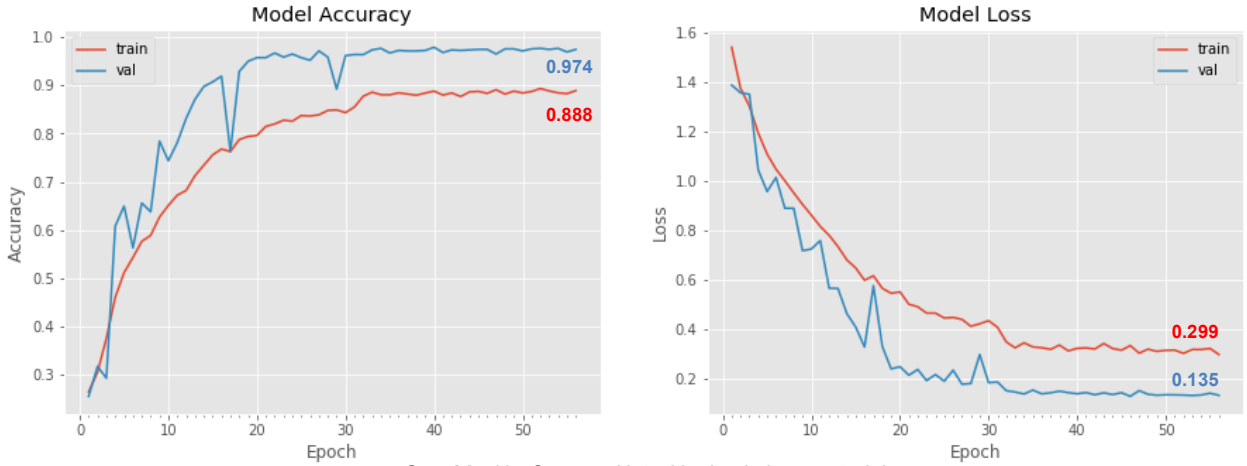
IV – Results

Model Evaluation and Validation

After several iteration of sensitivity studies, the final model selected is SqueezeNet with simple by pass. The model was trained from scratch with :

- Kernel initializer : he_normal
- Optimizer : Adam optimizer with learning rate 0.0001
- Loss function : Categorical corss entropy
- Training parameters : EarlyStop_callbacks, checkpointer and ReduceLROnPlateau
- Epoch : the training stopped at 56 epochs for 5008 images for training dataset and 968 images for testing dataset.

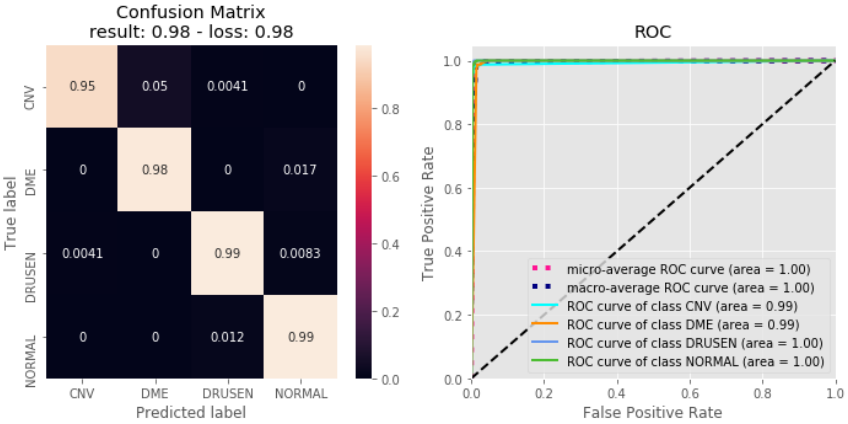
During the training, the model reached 97.4% accuracy with training dataset, 88.8% accuracy with testing dataset for validation. For the loss, it reaches 29.9% loss for the training dataset and 13.5% with testing dataset for validation (**Graphic 12**).



***Graphic 12** : SqueezeNet with simple bypass training*

Also, classification report shows good precision/recall/F1-score of the model for the ocular disease classification. Another way to visualize the classification performance is to use the confusion matrix, and ROC curve tools. Based on confusion matrix, the classification scores are 95% for CNV, 98% for DME, 99% for DRUSEN and 99% for NORMAL (**Graphic 13**).

	precision	recall	f1-score	support
CNV	1.00	0.95	0.97	242
DME	0.95	0.98	0.97	242
DRUSEN	0.98	0.99	0.99	242
NORMAL	0.98	0.99	0.98	242
accuracy			0.98	968
macro avg	0.98	0.98	0.98	968
weighted avg	0.98	0.98	0.98	968



***Graphic 13** : SqueezeNet with simple bypass model performance*

The size of the model save after the training is 9.1Mo.

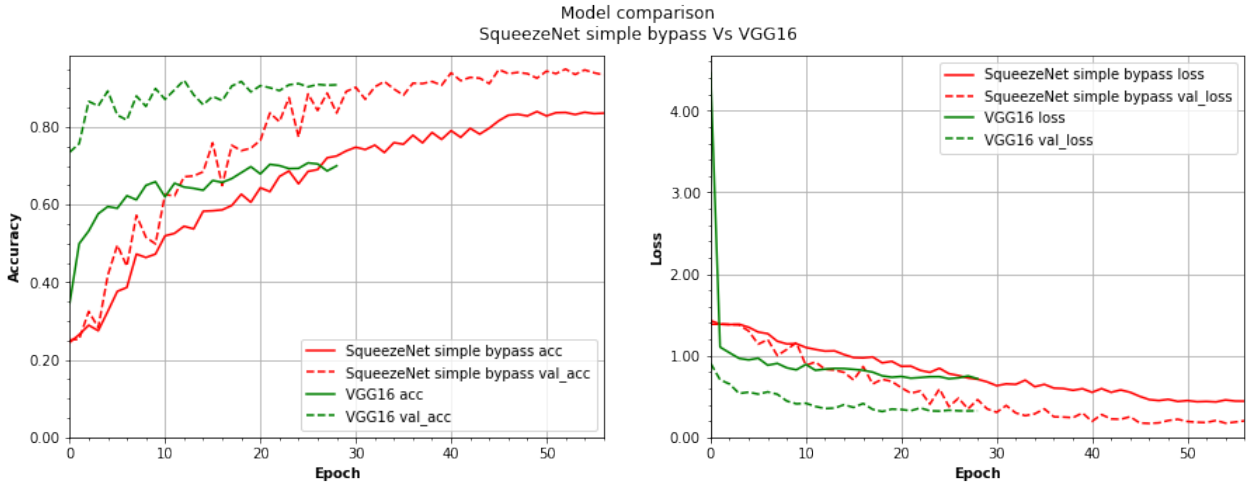
Justification

Based on the benchmark (see Benchmark paragraph), 2 aspects are focused :

Training performance

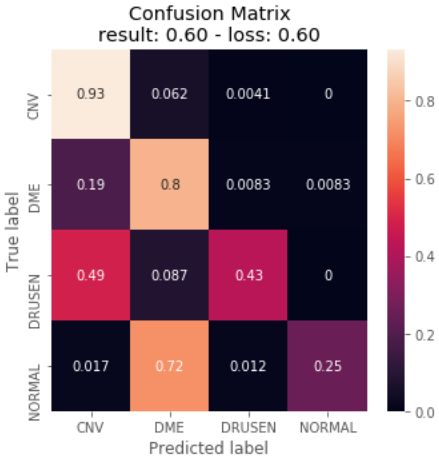
Instead of reproducing the training of VGG16 pretrained model (or other) described in benchmark paragraph, due credit limitation on AWS, a model using pretrained VGG16 was created with similar model parameter (except rescaling 1/255 for VGG16 and feature wise center/stdNormal for SqueezeNet simple Bypass) and trained with smaller number of images, in order to have comparison elements.

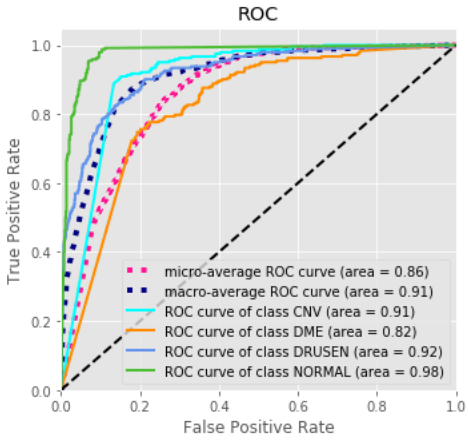
For training performance, SqueezeNet with simple Bypass model needs more time for training to reach an accuracy and loss plateau, while VGG16 pretrained model is quicker (**Graphic 14**).



***Graphic 14** : Training performance - SqueezeNet with simple bypass Vs VGG16*

	precision	recall	f1-score	support
CNV	0.58	0.93	0.71	242
DME	0.48	0.80	0.60	242
DRUSEN	0.94	0.43	0.59	242
NORMAL	0.97	0.25	0.40	242
accuracy			0.60	968
macro avg	0.74	0.60	0.57	968
weighted avg	0.74	0.60	0.57	968





***Graphic 15** : Classification performance - SqueezeNet with simple bypass Vs VGG16*

Model size and time prediction

Although the training for VGG16 model sample created and SqueezeNet with simple Bypass were not made in the fully same conditions, the size of the model generated can give an indication about the compression rate obtain with SqueezeNet model. The pretrained VGG16 model file is 136Mo, while the SqueezeNet model file is 9.1Mo. In terms of time computation for prediction, VGG16 takes 9 seconds and SqueezeNet 3 seconds to predict ocular diseases with 32 OCT scans.

V – Conclusion

Free-Form Visualization

To test the model, prediction are made with few OCT scans. It shows good prediction for the classification (**Figure 7**), except for CNV case, which is coherent because of the 95% accuracy observed in **Graphic 13**.

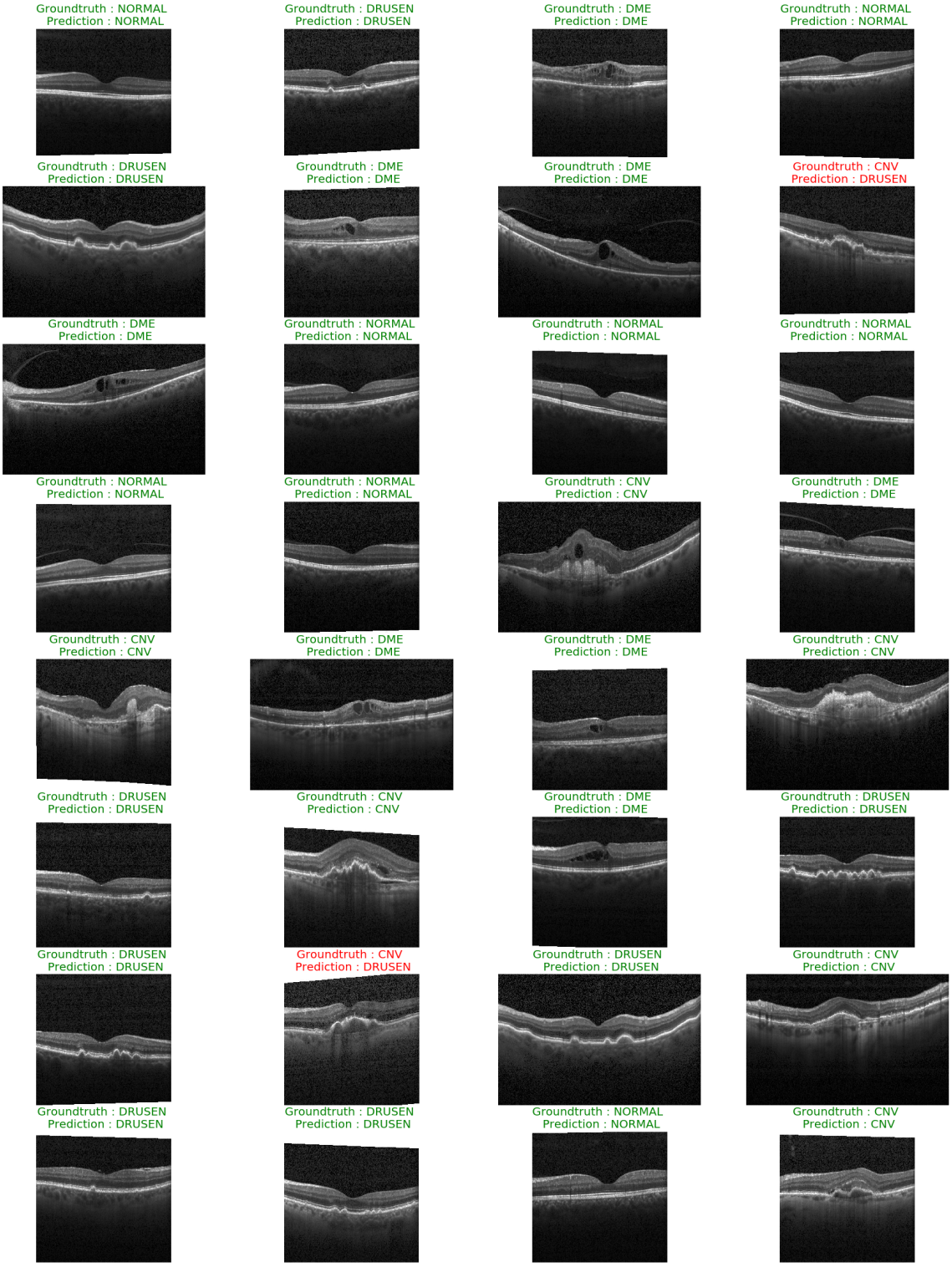


***Figure 7**: Classification prediction test - SqueezeNet with simple bypass*

Reflection

The capstone can be summarized in following steps :

1. Get good and enought dataset to start the project.
2. Setup AWS EC2 service for calculation
3. Create a benchmark as reference
4. Analyze the dataset
5. Sensitivity studies for data preprocessing
6. Multiple training and evaluation SqueezeNet model for sensitivity studies
7. Train and evaluate the best SqueezeNet model setup
8. Prediction test on few dataset

From the begining to the end of this capstone, steps 2 and 5+6 were the most difficults, but interesting and challenging for investigations.

For step 2, I had to familiarized to AWS environment and setup environment to enable GPU.
And for steps 5+6, lots of tries and errors were done to get a correct model trained as a starting point for the improvement step.

At the end, the SqueezeNet model with simple Bypass fit the initial expectation. It has a good performance for classification, and the file saved after the training is very small.

However, although the proposed model does correctly the ocular disease analysis with all its advantages (size, time prediction), the training performance was obtained under image preprocessing conditions and computational time. That is not the case for VGG16 or other pretrained model, which show good training performance with whatever preprocessed image quality (« in a certain limit ») and a good classification performance after few epochs.

Improvement

Because of the nature of the problem, accuracy needs to be improved. Many option can be investigated :

- A more refine grid search parameter/architecture can be done on :
    - Squeeze ratio in SqueezeNet model (Change the number of filter in fire module)
    - Number of fire module in SqueezeNet model
- Test other model like Transformer-XL as proposed in the capstone proposal review.
- Do a segmentation on OCT scans (like U-Net), in order to identify different ocular layers and train the model with segmentation data.

Then another aspect that would be interesing to improve is the time for training. If new type of dataset needs to be implemented in the model, it is always to interesting to have a new model in a minimum of time.