OCT DEMO

(PyTorch Implementation)

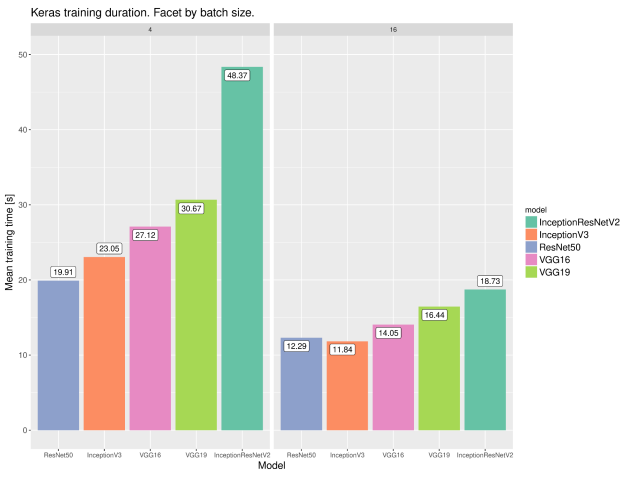
Introduction

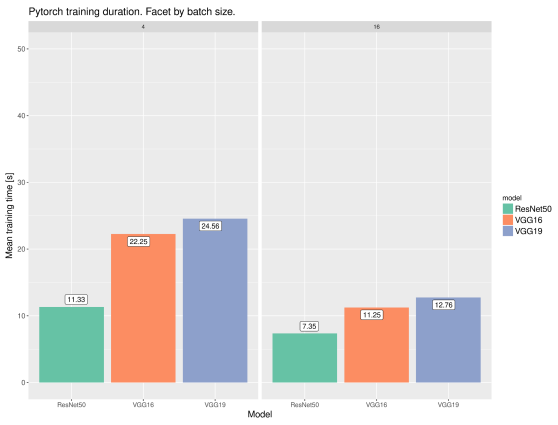
The project aims to build a CNN, capable of classifying medical images concerning OTC.
Then create a help and supervision system that supports doctors in diagnosing the problem.
In this project various models will be used and then compared to arrive at performances that can be used in real environments.
Unfortunately, having initially limited resources, restrictions on training time will apply, in order to identify the model that performs best and then train it appropriately, as soon as all the bugs have been identified and eliminated.

Why PyTorch

initially keras was used, but we immediately ran into the problem of limited performance, especially if we think of this problem as very scalable.

PyTorch, besides being much faster and suitable in situations with large datasets, also allows for more meticulous architecture programming and debugging with higher capabilities. Below are the performance graphs of keras vs PyTorch.



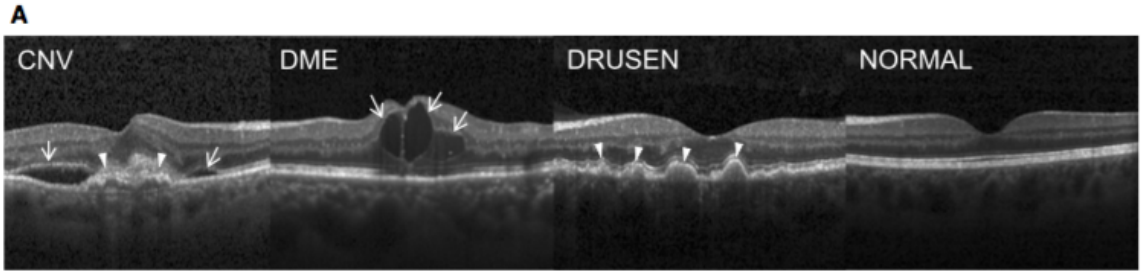


Dataset

[Retinal OCT Images (optical coherence tomography) | Kaggle]

The dataset contains about 85k images, divided into training, test and validation set folders. The classes within the dataset are 4: normal, drusen, CNV, DME Example of Data for each class .

Example of Data for each class :



All the images in the dataset are in black and white , so only one channel.

All images have been uploaded to Google Drive, in order to allow Google Colab to access them directly.

Load Phase

Created from dataset stored in Google Drive , three portion of data (Train , Test , Validation).

Same steps for load a dataset, done also by the official website of PyTorch :

1. **Data_transform** : define the transformation which will be applied to image , here is possible make augmentation
2. **Image_dataset** : apply the transformation
3. **Dataloaders** : create the stream of data in portion of batch size

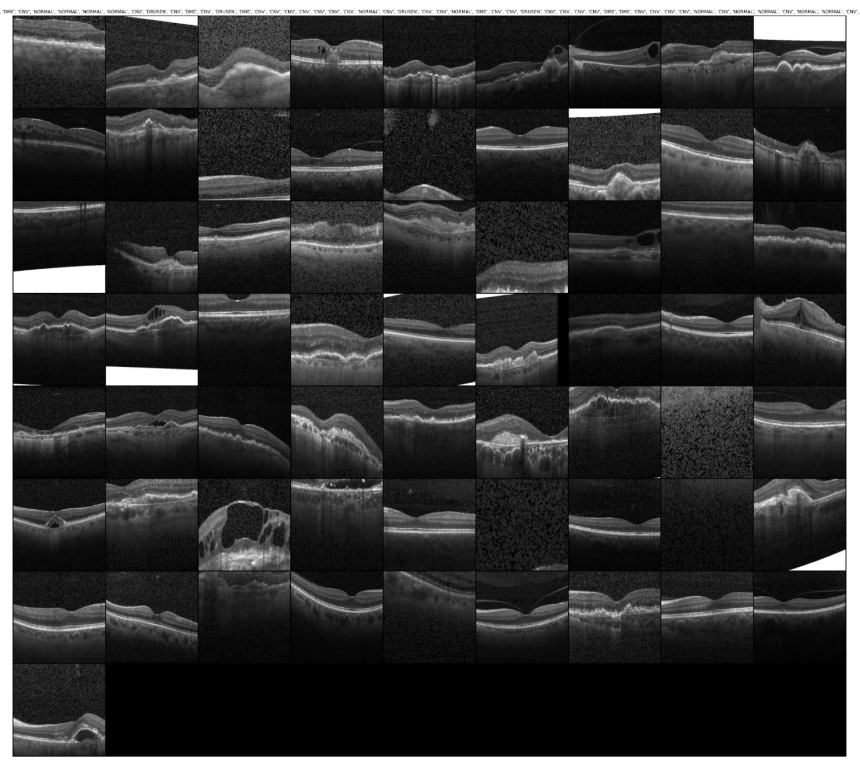
The *input* has been set to 224x224 , because it is the input quantity to the ResNet18 network.

The *augmentation* should be done in the create step of data_transforms , but in this case it seems to be useless for now since we have enough samples for a first demo. An augmentation will be created in the near future.

```
data_transforms = {
    TRAIN: transforms.Compose([
        # Data augmentation is a good practice for the train set
        # Here, we randomly crop the image to 224x224 and
        # randomly flip it horizontally.
        transforms.RandomResizedCrop(224), #fissato una size di 224
        transforms.RandomHorizontalFlip(),
        transforms.ToTensor(),
    ]),
    VAL: transforms.Compose([
        transforms.Resize(256),
        transforms.CenterCrop(224), #-> per resnet va bene 224
        transforms.ToTensor(),
    ]),
    TEST: transforms.Compose([
        transforms.Resize(256),
        transforms.CenterCrop(224),
        transforms.ToTensor(),
    ])
}
```

The batch size is setted to 64 , this value is optimal for the training of image classifier , and it's used also to speed up the creation of model.

Below we have an example of batch , used also in the training :



Architecture Used

ResNet18 - Limited Training

Description :

The ResNet-18 architecture consists of 18 layers, including convolutional layers, pooling layers, and fully connected layers. The network takes as input an RGB image of size 224x224 and produces an output of 1000 classes.

One of the key innovations of ResNet-18 is the use of residual connections, also known as skip connections, which allow information to bypass one or more layers in the network. This helps to alleviate the vanishing gradient problem and allows the network to learn deeper and more complex representations.

In our case we used Resnet but using the transfer learning technique specifically the Feature reuse.

The **Feature Reuse** serves to adapt our problem to the network, this technique was used because it is more effective than the Fine Tuning in the event that the dataset differs from the dataset used in the training of the original network, in this case our dataset is composed of images medical datasets, which differ greatly from the ImageNet dataset used for ResNet training18.

The network has therefore been frozen , so as not to be able to modify the weights within the model , and then a classifier has been attached to it which outputs the 4 classes of our problem.

Training

As already mentioned, for this computation a training limited to 4 epochs was used, with a _step_per_epoch = 100. Therefore only a small part of the training set (Batch_size*100 = 6400 images) is used, out of a non-Augmented total of 80k .
So we expect , suboptimal , but quick results to train the network.

If available , the model is moved to the Gpu , for faster computation.

```
if use_gpu:
    model_conv.cuda() #.cuda() will move everything to the GPU side
```

Loss function in PyTorch that is commonly used for multi-class classification tasks. It combines nn.LogSoftmax() and nn.NLLLoss() in a single class to make it more computationally efficient.

```
criterion = nn.CrossEntropyLoss()
```

Train_model

To the 'train_model' function, useful modifications have been added to track the training progress, such as the remaining time for each epoch, a progress bar for the steps, and data for the loss of the validation/train and accuracy of the validation/train , useful for evaluating overfitting/underfitting

Early Stopping :

Early stopping , with patience equal to 5.

Equivalent to early stopping 'Max' i.e. in "max" mode it will stop when the quantity monitored has stopped increasing. The monitored quantity is Epoch Accuracy.

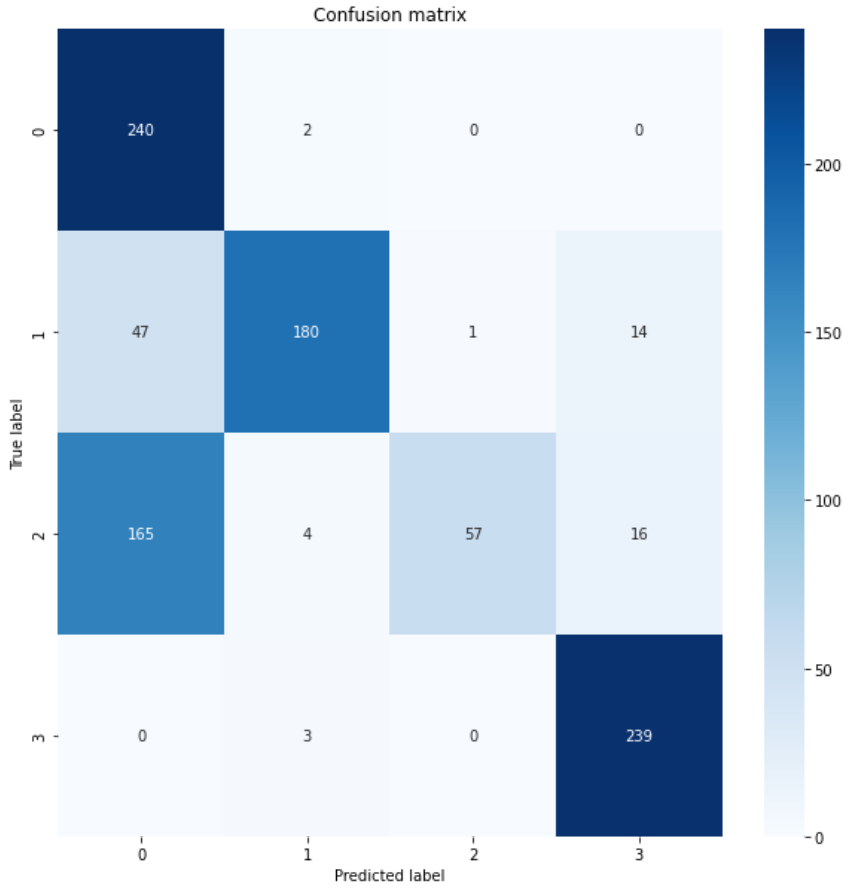
Test & Result

Test runned on entire Training Set

Classification Report

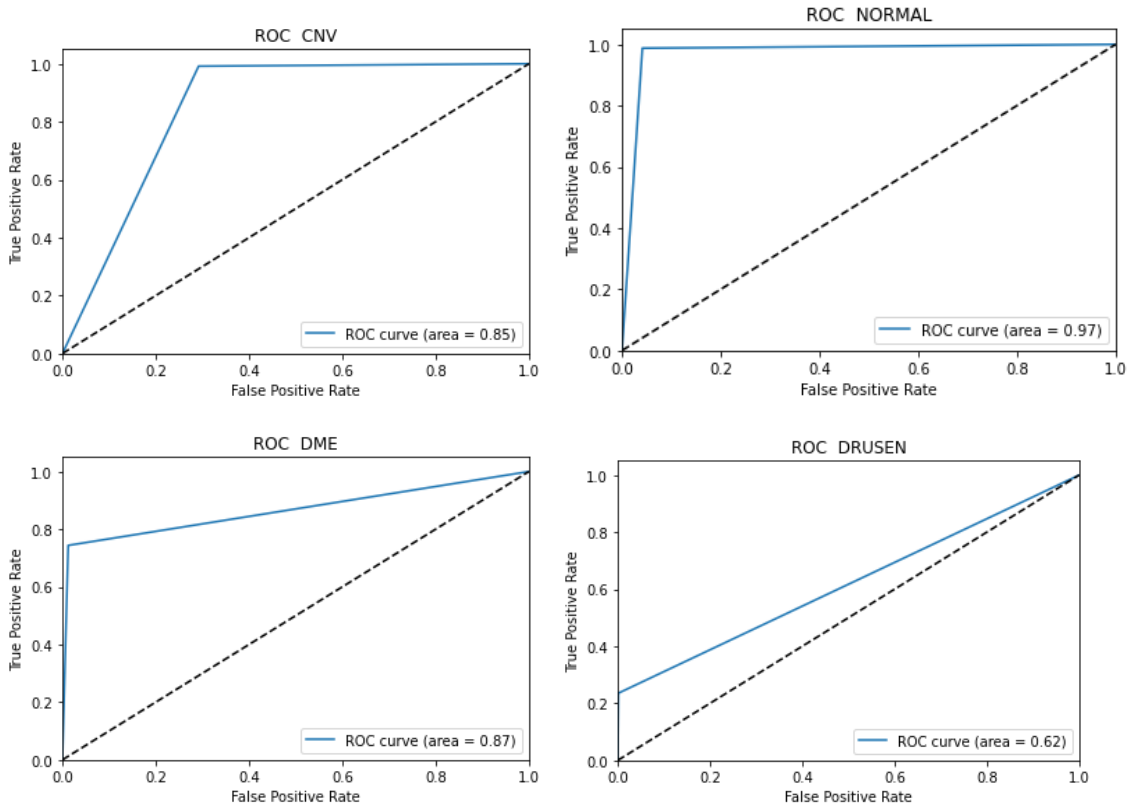
```
Classification report:
              precision    recall  f1-score   support

           0       0.53      0.99      0.69       242
           1       0.95      0.74      0.84       242
           2       0.98      0.24      0.38       242
           3       0.89      0.99      0.94       242

    accuracy                           0.74       968
   macro avg       0.84      0.74      0.71       968
weighted avg       0.84      0.74      0.71       968


Evaluation completed in 1m 44s
Avg loss (test): 0.0106
Avg acc (test): 0.7397
```

Confusion Matrix

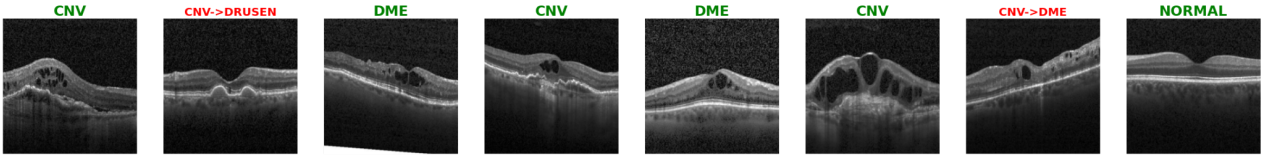


Roc curve

We can see that there is a problem in the roc curve concerning the DRUSEN , the area is only 0.62 , this problem can also be seen in the classification report and in the confusion matrix.
Obviously having been trained so little, the situation is normal, but it will have to be investigated if the problem persists, in more massive training.



Graphical example on some image



Technique Used

The techniques used to counter overfitting are:

- Early Stopping
- Augmentation

Conclusion

The model seems well done, so **it's possible to proceed with a massive training** , taking advantage of the entire Training set and also applying the augmentation.
However, we need to understand the behaviour of the Drusen class, which is currently confused with other classes.