Retinal Optical Coherence Tomography Images Classification

Abstract

In this project, we use CNN transfer learning to classify retinal damage from OCT scans. What we will do is using a model (VGG-16 and GoogLeNet) which is already capable of extracting features from an image and train its fully connected network in order to classify different types of retinal damage instead of objects. Also we try to improve its performance by using some machine learning algorithms and adjusting parameters. Finally we will reach to an overall graph based analysis and evaluation for this model. This work will use [PyTorch](https://pytorch.org/) as deep learning framework and [CUDA](https://developer.nvidia.com/cuda-zone) for GPU acceleration.

1 Introduction

1.1 About Retina OCT

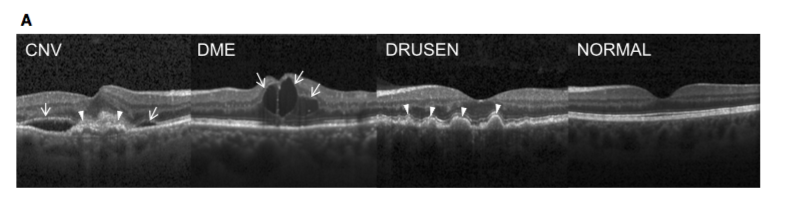
Retinal optical coherence tomography (OCT) is an imaging technique used to capture high-resolution cross sections of the retinas of living patients. Approximately 30 million OCT scans are performed each year, and the analysis and interpretation of these images takes up a significant amount of time (Swanson and Fujimoto, 2017)[1].

Figure 1. Representative Optical Coherence Tomography Images and the Workflow Diagram [Kermany et. al. 2018] <http://www.cell.com/cell/fulltext/S0092-8674(18)30154-5>

(A) (Far left) choroidal neovascularization (CNV) with neovascular membrane (white arrowheads) and associated subretinal fluid (arrows). (Middle left) Diabetic macular edema (DME) with retinal-thickening-associated intraretinal fluid (arrows). (Middle right) Multiple drusen (arrowheads) present in early AMD. (Far right) Normal retina with preserved foveal contour and absence of any retinal fluid/edema.[2]

1.2 Transfer learning and CNN

[Transfer learning](https://en.wikipedia.org/wiki/Transfer_learning) turns out to be useful when dealing with relatively small datasets; for examples medical images, which are harder to obtain in large numbers than other datasets. Instead of training a deep neural network from scratch, which would require a significant amount of data, power and time, it's often convenient to use a pretrained model and just finetune its performance to simplify and speed up the process.

In short, convolutional networks are used on (but not only) images instead of fully-connected feedforward networks because they would require a very high number of neurons - i.e. at least one per pixel in the input layer - and that would make them inconvenient.

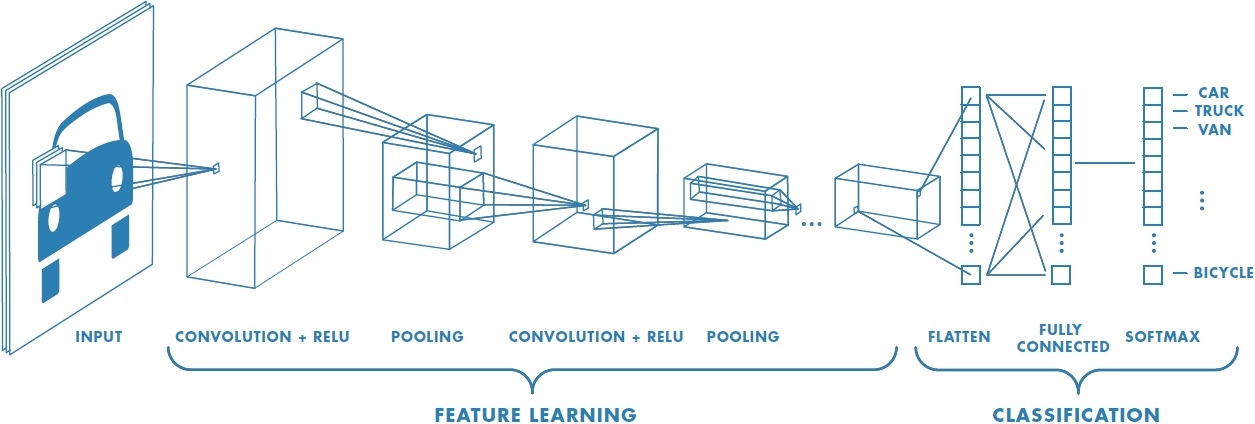
CNNs solve this problem by using a different architecture, summarized in Figure 2.

Figure 2: CNN architecture (from <https://it.mathworks.com/discovery/convolutional-neural-network.html>)[3]

We can identify two main blocks inside of a typical CNN:

* Feature extraction
* Classification

The feature extraction is made of a series of convolutional and pooling layers which extract features from the image, increasing in complexity in each layer. These features are then fed to a fully connected network (classifier), which learns to classify them.

2 Methods

Our project uses different CNN models for image classification and compared their performance. The models we choose are VGG-16 and GoogLeNet.

2.1 VGG-16

VGG is a convolutional neural network model proposed by K. Simonyan and A. Zisserman from the University of Oxford in the paper “Very Deep Convolutional Networks for Large-Scale Image Recognition” . The model achieves 92.7% top-5 test accuracy in ImageNet , which is a dataset of over 14 million images belonging to 1000 classes.

It is currently the most preferred choice in the community for extracting features from images. The weight configuration of the VGGNet is publicly available and has been used in many other applications and challenges as a baseline feature extractor.

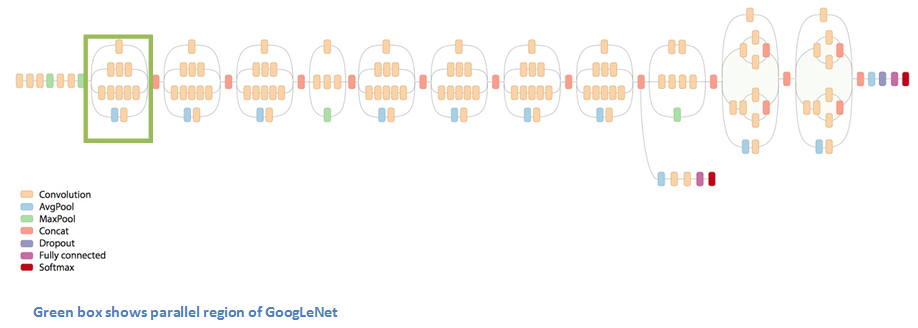
The macroarchitecture of VGG-16 can be seen in Fig. 3.

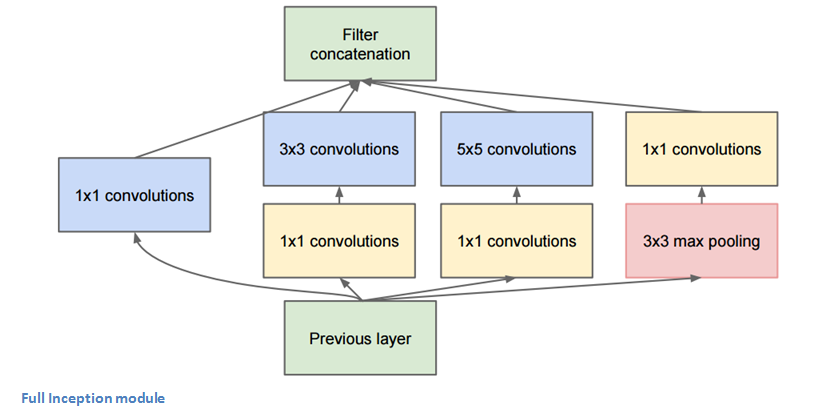
Figure 3: Macroarchiteture of VGG-16

The input of training VGG16 is fixed-size 224 x 224 RGB images, and the output is the classification results contains 1000 channels(one for each class). The image is passed through a stack of convolution layers with a very small 3 x 3 receptive field. A 1x1 convolution filters is also used in one of the configurations and can be seen as a linear mapping of the input channels. The task of spatial pooling has been carried out using five max-pooling layers, which follow some of the convolutional layers but not all the convolutional layers.

2.2 GoogLeNet

GoogLeNet is a 22 layer CNN and was the winner of ILSVRC 2014 with a top 5 error rate of 6.7%.



The green box is called **Inception module**: apply filters of different sizes in parallel and concatenate their answers. Each module captures in parallel details at different scales and has thus different effective receptive fields.

**Deeper with fewer parameters**: introduced “1 × 1” convolutions for dimensionality reduction. The first version had 22 layers and 4 million parameters.

3 Experiment

3.1 Dataset

Dataset of validated OCT and Chest X-Ray images described and analyzed in "Deep learning-based classification and referral of treatable human diseases". The OCT Images are split into a training set and a testing set of independent patients. OCT Images are labeled as (disease)-(randomized patient ID)-(image number by this patient) and split into 4 directories: CNV, DME, DRUSEN, and NORMAL.

3.2 Dataset loader

The dataset is divided in three categories: training, validation and test.

The first one will be, obviously, used for trainig; the validation set will be used to measure the model performance during training and the test set will be used to evaluate our model performance once the training has finished.

*Note:* These three sets should all contain different images.

Loading this dataset with pytorch is really easy using [ImageFolder](https://pytorch.org/docs/master/torchvision/datasets.html#imagefolder) as the labels are specified by the folders names.

3.3 Training

For every epoch we iterate over all the training batches, **compute the loss , and adjust the network weights.** Then we evaluate the performance over the validaton set. At the end of every epoch we print the network progress (loss and accuracy). The accuracy will tell us how many predictions were correct.