First Challenge "Solve classification problem"

Pierluigi Carlino team GNP[†]

ABSTRACT. Trough this document will be described all the attempts and experiments used to solve the following problem: Create a model that classify the image of a plant into one of eight possible species.

Keywords: CNN, Deep Learning,Classification Problem

APPROACHES USED

To solve the problem were used different approaches described in the following pages, that had as immediate consequence the development of some sort of basic functions to do experiments. These functions were build in order to make attempts in the easiest and fastest way possible considering that I found useful to cut all unnecessary and repetitive tasks.

In the attached notebook a further description of these modules are given.

PRE-PROCESSING AND AUGMENTATION

Basically a pre-processing pipeline was developed where each stage take as input an image and produce different images that were passed as input to the stage that follows.

Manly these are the used pipeline stages

- A Rotation stage: given an image and a number of steps n it produces n images rotated of 360/n degrees
- A Subdivision stage used to split an image in chunks
- A Blur stage used to add some noise to the original image using incremental kernels
- A Flip stage used to produce flipped images
- A Scaling stage used to scale the given image

Seven pipelines was used to do experiments

- (1.1) Rotation Stage only
- (1.2) Scaling Stage only(with different factors)
- (1.3) Subdivision Stage only
- (1.4) Rotation Stage(with a fixed angle step of 45 degrees),Scaling Stage (with different factors)
- (1.5) Subdivision Stage ,Scaling Stage
- (1.6) Subdivision Stage ,Rotation Stage(with a fixed angle step of 45 degrees),Scaling Stage(with different factors)

[*]pierluigi.carlino @mail.polimi.it

- (1.7) Subdivision Stage,Rotation Stage(with a fixed angle step of 45 degrees),Flip Stage,Blur Stage Scaling Stage (with different factors)

  The best results will be obtained with the (1.7) pipeline that produces around 10^6 images

KIND OF MODEL USED

In order to analyze the different results, attempts could be grouped in the following macro areas:

- (2.1)Big CNN networks (networks with more than 20 CNN layers each of them with more than 100 filters).
- (2.2)Small CNN networks (networks with less than 4 CNN layers each of them with less than 100 filters).
- (2.3)Medium CNN networks (networks with less than 10 CNN layers each of them with less than 150 filters).
- (2.4)Ensemble CNN networks obtained concatenating the best macro-model of (2.3) and (2.2)
- (2.5)Machine learning classifiers that have as input the output of a pre-trained macro-models ' hidden layer

TRAINING AND TESTING STAGE

Due to the amount of data that goes out from the augmentation pipeline I decided to use a sort of k-fold cross validation for training and testing.

In fact the training and testing stage was performed using the LearningStage.perform function which

- Initialize data-set reading at most a number of image specified by LearningStage.limit variable
- Start 3 iteration of the following instructions
  - Choose if the data-set must be equal(if all classes must have the exact number of samples)
  - Chose a number of epochs between LearningStage.baseEpoch and LearningStage.baseEpoch*LearningStage.epochIn (both multiplied by 0.1 if equal class data-set generation is picked)
  - Evaluate the model (randomly split data-set in train and test using scikit-learn train_test_split function)
  - Retrain the model (randomly split data-set in train and test using scikit-learn train_test_split function)
  - Save the model
- Call the LearningStage.updateVariables which for the moment is a simple empty function but it can be extend to perform some Genetic Algorithm's mutation function of the model

MODEL CONSIDERATION

(2.1)BIG CNN NETWORKS

During the training of these kind of CNN I noticed that the accuracy and the loss constantly improve each epoch, but slowly. Changing the dropping rate between the layers helped but not so much and this behavior is independent regardless the choice of data-set generation pipeline.

For that reason and the fact that the model is too big and considering that during the experiments there was not the possibility of a good internet connection for the model submitting I decided to leave this approach.

(2.2)Small CNN networks

Respect the (2.3) macro-model this macro-model is not so tiny but is faster to converge especially if it was meant to classify not all classes but an aggregation or a subset of them. I notice that even if on train data it starts to have a good accuracy and a good loss , on test set it has a 0.60-0.70 range for the accuracy and a 1.2-0.9 loss range. For this reason I started to merge the best of them in the (2.4) macro-model

(2.4)Ensemble CNN networks

Under the assumption that the models of type (2.2) learn a small idea/concept from the image I decided to concatenate them using a common layer as output which is followed by hidden layers. Then after creating these new models (using the mergeModels function in covNet module) i started training them. I notice that the loss range of (2.2) persists while the accuracy range changed into 0.80-0.90 on test data-set.

(2.3)Medium CNN networks

It seems to be really similar to (2.2) despite the fact that is faster to converge. Despite all of previous macro-models it seems to not develop a bias. For bias I meant some preference to predict a class respect to others. In fact the previous macro-models for same classes never predict them, and during the submission of the model this leads to the classic error of "F1 undefined"

Last consideration

Finally for the lack of time and resources I can't deep explore the macro model (2.3) because it takes to time to train even if it improves epochs after epoch.

So I trained the 2.2 macro-model obtaining good results that I found to be in line with the ones that came out from submissions (in fact the accuracy on test-set was very similar to the ones obtained by submissions)