Dipartimento di Ingegneria Informatica
Automatica e Gestionale Antonio Ruberti

Sapienza
Università di Roma

Machine Learning: homework 2

Image Classification with Small Data

Student:
Federica Cocci

ID:
1802435

December 23, 2021

Contents

1 Problem to be resolved

This homework has been assigned during the seminars "Deep Ensembles" and "Tune It or Don't Use It: Benchmarking Data-Efficient Image Classification".
Generally speaking the problem I have faced up is a classification problem having a small dataset.
We know that a large amount of training data plays a critical role in making the models successful. ResNet, an image classification architecture, won the 1st place in the ILSVRC 2015 classification competition with ∼50% improvement in the previous state of the art. This ResNet not only had a very complex and deep architecture but was also **trained on 1.2 millions of images**.

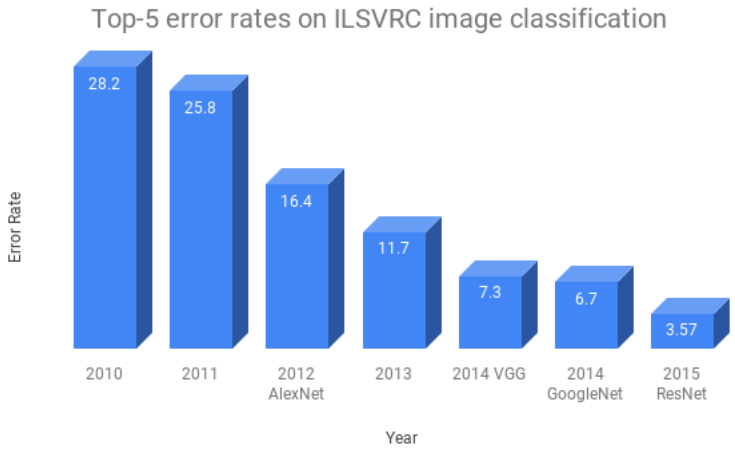


Figure 1: ILSVRC top model performance across years

1.1 How large datasets help in building better Machine Learning models?

We generally want to minimize both bias and variance so we want to build a model which not only fits the training data well but also generalizes well on test/validation data. There are a lot of techniques to achieve this but training with more data is one of the best ways of achieving this.

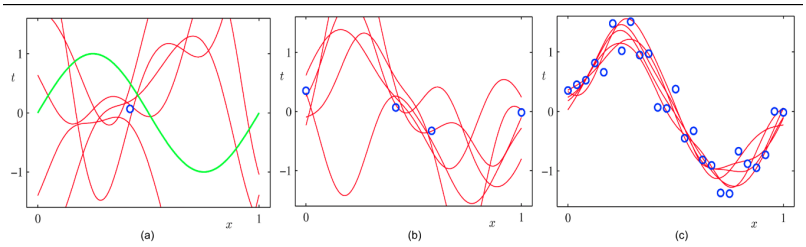


Figure 2: Large data results in better generalization

Let's say we have a data distribution which is similar to a sinusoidal wave. Fig(2a) depicts that multiple models are equally good in fitting the data point. A lot of those models overfit and do not generalize well on the whole dataset. As we increase data, Fig(2b) illustrates a reduction in the number of models which can fit the data. As we further increase the number of data points, we successfully capture the true distribution of the data as shown in Fig(2c).

1.2 How to resolve the problem of less data?

Actually because deep learning model performance depends on data size we have to understand how to work with smaller data sets to get similar performances.
Having less data leads us to some consequences such as lack of generalization, data imbalance, difficulty in optimization. Let's analyze how to face up these implications:

- Lack of generalization can be resolved using for example:
  - Data Augmentation
  - Ensemble
- Data Imbalance can be resolved using for example:
  - Change the loss function
  - Balancing the dataset
- Difficulty in optimization can be resolved using for example:
  - Transfer Learning
  - Problem reduction
  - Learning with less data

Deep neural networks have millions of parameters to learn and this means we need a lot of iterations before we find the optimum values. In order to influence the optimization process, there are some factors we can work with such as optimization algorithm and hyperparameters initialization.
There are some techniques which are more relevant when we work with a deep neural network and a small dataset and here there are some examples:

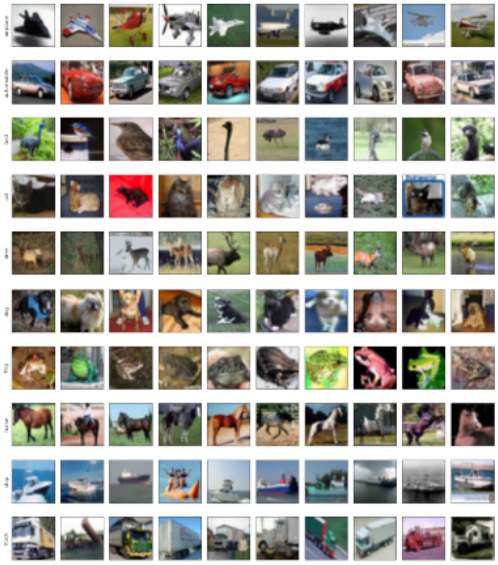
- Transfer learning
- Meta-Learning
- Data Augmentation
- Data Generation

1.3 What is our approach to the problem

Our goal is to build a Data-Efficient Image Classification having limited data, without using pre-trained networks and, using as basic architecture the Residual Network (ResNet), playing with hyperparameters, customizing a network architecture or changing the optimizer.

2 Dataset

The dataset used for this experiment is ciFAIR-10 that is a variant of the popular CIFAR dataset: in CIFAR dataset lots of images in the test set have duplicates in the training set and these duplicates may bias the image recognition techniques which have no more the generalization capability; to eliminate this bias, the "fair CIFAR" (ciFAIR) dataset is provided and here all duplicates in the test sets are replaced with new images sampled from the same domain.
It is made of RGB images of size 32x32 from 10 classes of everyday objects: airplane, automobile, bird, boat, cat, deer, dog, frog, horse, truck.



(a) Some samples from the 10 classes

Split	Total Images	Images / Class
train	300	30
val	200	20
trainval	500	50
test	10,000	1,000

(b) splits of the ciFAIR-10

Figure 3: ciFAIR-10

3 Architectures

Let's here analyze the two basic architectures we work with.
When we increase the number of layers in a deep neural networks, there is a common problem in deep learning that is called Vanishing Gradient.

3.1 ResNet

Residual Network, also called ResNet, was proposed in 2015 by researchers at Microsoft Research. In order to solve the problem of the vanishing gradient, this architecture makes use of residual blocks to improve the accuracy of the models. The concept of "skip connections," which lies at the core of the residual blocks, is the strength of this type of neural network.
These skip connections work in two ways:

- They alleviate the issue of vanishing gradient by setting up an alternate shortcut for the gradient to pass through.
- They enable the model to learn an identity function.

This ensures that the higher layers of the model do not perform any worse than the lower layers.

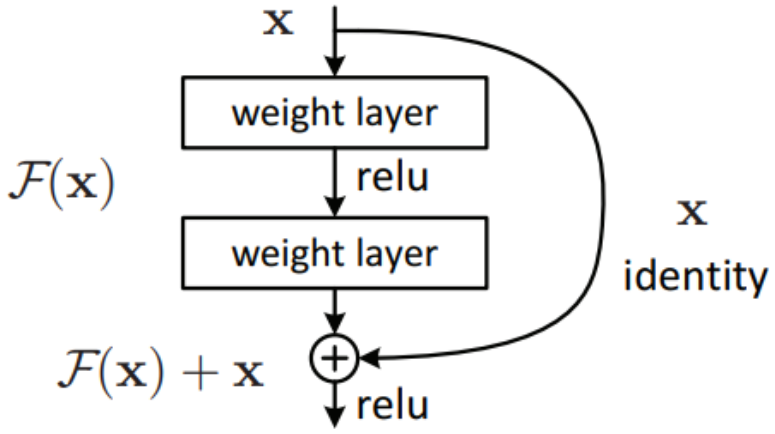


Figure 4: Residual block

Thanks to the residual blocks, ResNet improves the efficiency of deep neural networks with more neural layers while minimizing the percentage of errors, making it possible to train much deeper networks than previously possible.
ResNet has many variants that run on the same concept but have different numbers of layers. For example Resnet50 is used to denote the variant that can work with 50 neural network layers.

3.2 Wide ResNet

Wide Residual networks simply have increased number of channels compared to ResNet; otherwise the architecture is the same.
Wide ResNet has many variants that run on the same concept but have different numbers of layers plus different numbers of channels. For example wide_resnet_50_2 is used to denote the variant that works with 50 neural network layers and 2 channels. In other words in a Wide ResNet we can change the depth and the width.

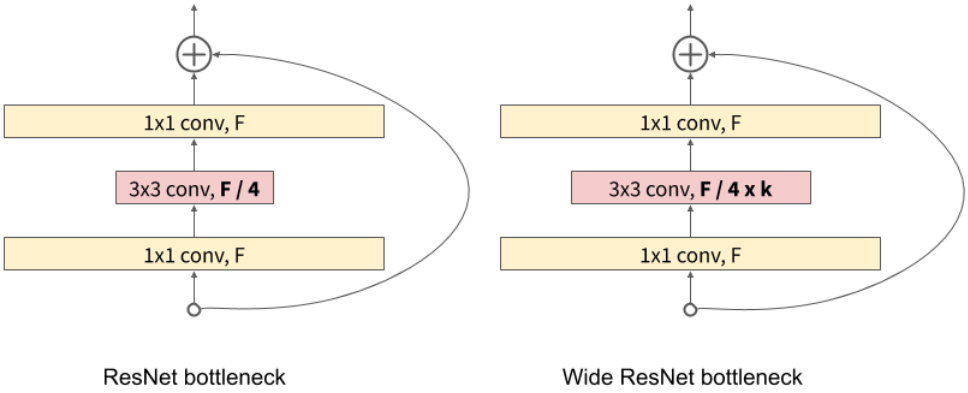


Figure 5: ResNet VS Wide ResNet

4 Optimizers

Optimizers are methods used to modify the attributes of the neural network such as weights and learning rate to reduce the losses and in order to minimize as much as possible the function to solve the optimization problem. There are several optimizers, for example:

- Gradient Descent (GD)
- Stochastic Gradient Descent (SGD)
- Adagrad
- AdaDelta

What I used to make experiments and also for the final model, are Stochastic Gradient Descent and AdaDelta.

4.1 Stochastic Gradient Descent

It's a variant of Gradient Descent. Actually "stochastic" stands for "random". So now the question is "where the randomness is introduced in the algorithm?": while selecting data points at each step to calculate the derivatives. In fact SGD randomly picks one data point (or a batch) from the whole data set at each iteration to reduce the computations enormously. SGD moreover tries to update the model's parameters more frequently and this thing leads the model to converge in less time.

In other words, while Gradient Descent runs through all the samples of the training set to do a single update for a parameter in one iteration, the Stochastic Gradient Descent uses only one or a subset (in this case it is called Minibatch Stochastic Gradient Descent) of training samples from the training set to do the update for a parameter in one iteration.
SGD often converges much faster compared to GD but the error function is not as well minimized as in the case of GD.

4.1.1 Momentum

Because SGD can largely vary through the iterations then the momentum is introduced to accelerate learning. The momentum accelerates the convergence towards the relevant direction and reduces the fluctuation to the irrelevant direction.

4.2 Adagrad

One of the disadvantages of GD and SGD is that the learning rate is constant for all parameters and for each cycle.
Adagrad changes the learning rate. It changes the learning rate 'μ' for each parameter and at every time step 't'.
For this optimizer, the manually tuning of the learning rate is not needed but on the other hand, because the learning rate is always decreasing, this results in a slow training. In the Adagrad, we store the sum of the squares of the gradients up to time step 't'. For this reason AdaDelta is introduced.

4.2.1 AdaDelta

Instead of accumulating all previously squared gradients, AdaDelta limits the window of accumulated past gradients to some fixed size w.

5 My solution

My approach to the problem has been first making some experiments (see the table below on paragraph 5.1) running only for 20 epochs changing architecture and/or optimizer and/or hyperparameters.
I decide to work with a Wide ResNet, in particular the wrn-22-8, and also a customized version of the same architecture that I called wrn-22-8-bho.

As optimizer I used the Stochastic Gradient Descent and an adaptive optimizer, the AdaDelta.

5.1 Table of experiments

Experiments List			
Architecture	**Optimizer**	**Hyperparameters**	**Accuracy**
wrn-22-8	AdaDelta	rho=0.9 eps=$1e^{-06}$ lr=$3e^{-3}$	22.31%
wrn-22-8	AdaDelta	rho=0.9 eps=$1e^{-06}$ lr=$4.55e^{-3}$	25.83%
wrn-22-8	AdaDelta	rho=0.9 eps=$1e^{-06}$ lr=$5e^{-3}$	26.71%
wrn-22-8	SGD	momentum=0.9 lr=$5e^{-3}$	44.32%
wrn-22-8-bho	AdaDelta	rho=0.9 eps=$1e^{-06}$ lr=$5e^{-3}$	28.26%
wrn-22-8-bho	SGD	momentum=0.9 lr=$6e^{-3}$	43.75%
wrn-22-8-bho	SGD	momentum=0.9 lr=$5e^{-3}$	45.19%

Analyzing the table:

- both the architectures work better with an high learning rate (testing 3 different values for the learning rate).
- between wrn-22-8 and wrn-22-8-bho, the one which gives me the higher accuracy (on 20 epochs) is the second one.

5.2 Final models

At the end of these experiments on 20 epochs, I have decided to use the customized architecture wrn-22-8-bho and, because it was requested to do at least 2 final experiments, I run for 500 epochs:

1. wrn-22-8-bho + SGD
2. wrn-22-8-bho + AdaDelta

Both the models have the learning rate set to

$$lr = 5e^{-3}$$

and I have added some hyperparameters:

- for SGD, I have introduced the momentum

$$momentum = 0.9$$

- for AdaDelta, I have introduced

$$rho = 0.9$$

$$eps = 1e^{-06}$$

5.3 WRN-22-8-BHO

My architecture is from the provided wrn-22-8 to whom I have added some layers at the end of the architecture, before the last ReLu and the 2D adaptive average pooling. In order:

- Convolutional layer
- ReLu activation function
- Max Pooling 2D
- Batch Normalization
- Convolutional layer

6 Comparisons and final observations

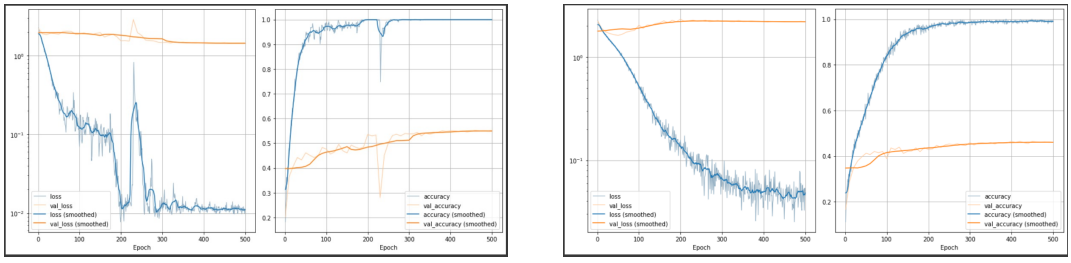
The goal of this homework is also obtain a decent accuracy in order, with further modifications, to beat the baseline below.

Baseline to beat:
- Wide ResNet-16-8 (wrn-16-8) , epochs = 500, bs = 10, lr = 4.55e-3, wd = 5.29e-3
- Accuracy on test set: 58.22%

Figure 6: Baseline to beat

The val_accuracy that I obtain with wrn-22-8-bho + SGD is 55.12%.

The val_accuracy that I obtain with wrn-22-8-bho + AdaDelta is 45.98%.



(a) wrn-22-8-bho + SGD (b) wrn-22-8-bho + AdaDelta

Figure 7: Accuracy

Consideration about wrn-22-8-bho + SGD:
It is clear from the graph above that the model overfits on the second half of the epochs.
The high oscillations may be caused by the optimizer itself because SGD oscillates to reach the minima but it can be noticed that at the end it converges.
Actually also modifying the momentum could have been a solution to all these oscillations.

Consideration about wrn-22-8-bho + AdaDelta:
It grows exponentially with less oscillations. I'm sure that if I have been started with an higher initial learning rate, AdaDelta would have benefit returning a higher accuracy that for the moment is below the 51%.

To avoid overfitting I could have used a network not so deep like the wrn-16-n

or maybe I could have applied regularization such as dataset augmentation, early stopping or dropout. Of course the overfitting is due also to the small dataset we have, so for sure it won't be avoided but it could be decreased.
All these possible solutions haven't been tested because of Colab resource and account limitations and because of time.

At the end the model I submit is the wrn-22-8-bho + SGD because is the model with the higher accuracy.