Technical Test Report : Image Classification

Elsa MARIE, engineering student at Grenoble-INP and KTH

October 16, 2019

1 Introduction

In an image classification context, our goal is to associate an image $\mathbf{X}$ with its class among a fixed set of classes. We worked on RGB images that come from the CIFAR10 dataset consisting on 60000 images of 10 classes. It is divided into 50000 training images and 10000 test images.

1.1 Problem Formulation

We consider a classification problem of C classes corresponding to C categories. Here, we have the 10 following categories : plane, car, bird, cat, deer, dog, frog, horse, ship and truck. Let D be the database available for us defined by a set of couples : $\{\mathbf{X}_i, c_i\}_{i=1}^{N}$ where $c_i \in \{1, ..., C\}$ describe the class (the category) to which the image $\mathbf{X}_i \in \mathbb{R}^{32\times32\times3}$ belongs.

The multi-class classification problem is defined by a function $f_\theta : \mathbb{R}^{32\times32\times3} \mapsto [0,1]^C$ such as $\forall i \in \{1, ..., N\},\ f_\theta(\mathbf{X}_i) = \mathbf{y}_i$. Each component j of the vector $\mathbf{y}_i$ can be seen as the probability that the image $\mathbf{X}_i$ belongs to the class $j \in \{1, ..., C\}$. It is defined by :

$$y_{i,j} = [\mathbf{y}_i]_{,j} = p(c_i = j|\mathbf{X}_i) \tag{1}$$

The purpose behind the classification model training is to evaluate the parameters θ that describe the function, such as for each couple i of the database, f_θ maximize the probability that the image belongs to the right class c_i and minimize the probability that it belongs to the others classes $c_{j\neq i}$. The class c_i de $\mathbf{X}_i$ can be estimated thanks to the model output $\mathbf{y}_i$ such as :

$$\widehat{c_i} = \underset{j\in\{1,...,C\}}{\text{argmax}}\ y_{i,j} = \underset{j\in\{1,...,C\}}{\text{argmax}}\ p(c_i = j|\mathbf{X}_i) \tag{2}$$

1.2 Performances criteria

To evaluate our classifier performances, we consider two good classification rates. Fisrt, we define the *Top1-Accuracy* rate as the rate of well-classified samples computed such as the higher component of the vector $\mathbf{y}_i$ is equal to the right class c_i. The *Top5-Accuracy* rate is defined such as the right class c_i belongs to the set of the five higher components of the vector $\mathbf{y}_i$.

2 Convolutional Neural Networks

2.1 Common Configuration

A typical neural network is composed by a certain amount of neurons. The output of a neuron is the result of an activation function $\sigma(.)$ applied to the linear combination of the inputs of the neuron with the set of parameters, usually called the *weights* of the network. Let x_i $\forall i \in \{1, ..., I\}$ the inputs of a neuron j, the ouput of this neuron is computed as following :

$$y_j = \sigma(\sum_{i=1}^{I} w_{ij}.x_i) \tag{3}$$

with w_{ij} the weight that connects the input i to the neuron j. A fully-connected layer of a neural network is composed by several neurons that are connected by a set of weights to all the neurons of the last layer.

Within a **convolutional** layer, all the weights are shared between the neurons thanks to some convolutional kernels. This type of layer is defined by four parameters : the number of kernels, the size of the kernel, the stride and the padding. A kernel acts as a filter on the image. The output of a convolutional layer is the result of the input image convolution with all the kernels of the layer. The center of the kernel is shifted according to the stride and the padding allows us to avoid the side effects.

The **pooling** layers are generally used with images because they allow us to reduce the size of the input. This compression is also made by convolutional kernels but those kernels are not defined by the weights of the network. The maxpooling layers take the maxima of a section of their inputs selected by the field of their kernels.

A convolutional neural network (CNN) is built with convolutional, pooling and full-connected layers. The weights of a CNN is defined by the set of the weights from the convolutional layers and the fully-connected layers.

According to [1], we build our classifier with a stack of convolutional and maxpooling layers followed by some fully-connected ones. We adapt the architectures provided by [1] to our own database CIFAR-10, whose images have a really small spatial resolution (32×32) in regards to those used in [1] (224×224).

However, we keep the same skeleton for our configurations. For the convolutional layer, we use filter kernels with a small size 3×3. The convolutional stride is fixed to 1 pixel and the padding is also 1 pixel in order to preserve the spatial resolution of the input image (preserve the same size). After two convolutional layers, we apply a spatial maxpooling that is performed by a 2×2 window with a stride of 2 pixels. Those three layers (conv. + conv. + maxpool.) describe a typical unit (*conv. unit*) for our different configurations.

After the stack of *conv. unit*, we apply two fully-connected layers with the respective sizes of 128 channels and 10 channels. The last one is a typical classification layer (softmax layer), whose the dimension is equal to the number of classes C. Except the last one, all layers are followed by a non linear activation function : the rectification ReLU defined by $\sigma : \mathbb{R} \mapsto \mathbb{R}^+$ such as :

$$\sigma(x) = \max\{0, x\}$$

The softmax layer gives us the output of the network : a C-dimensional vector, whose each component j describes the probability that the input belongs to the class $j \in \{1, ..., C\}$. By definition of a probability, each component has to belong to $[0, 1]$ and the sum of all components has to be equal to 1. According to this definition, the activation function of a classification layer is called *softmax* and is defined by $\sigma : \mathbb{R}^C \mapsto [0, 1]^C$ such as :

$$\sigma(y_{i,j}) = \frac{\exp(y_{i,j})}{\sum_{k=1}^{C} \exp(y_{i,k})} \mid j \in \{1, ..., C\}$$

.

2.2 Study of different configurations

In this report, I will study three different configurations that are based on those described in [1] but adapted to the CIFAR-10 dataset. By training those three models, I will study the influence of the CNN depth over the performances of classification. The architectures of CNN A, CNN B and CNN C are respectively defined by a stack of two *conv. units*, three *conv. units* and four *conv. units* as the depth of the architecture will be the more important difference between those three configurations. We provide a more detailed description with the following figure. Finally, it is important to notice that the number of the filters used in the convolutional layers increase with the depth of the architecture.

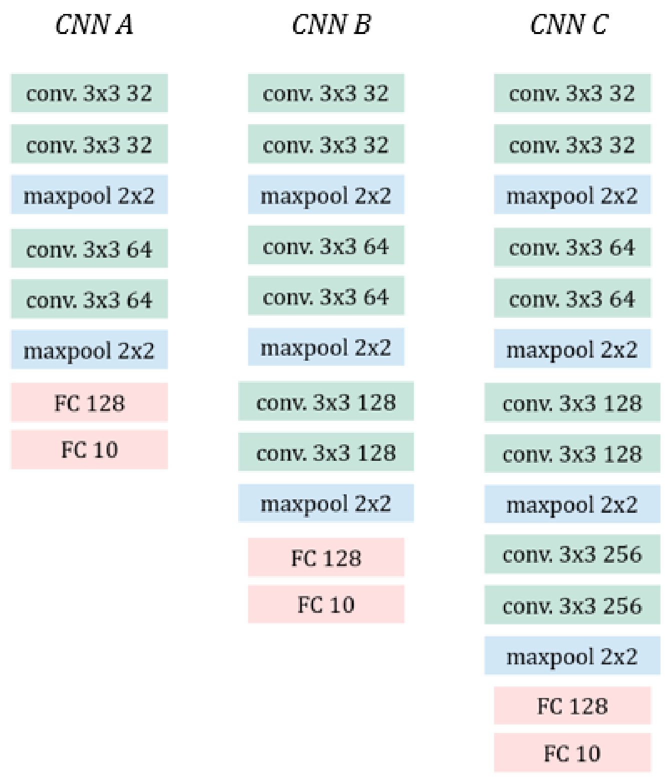


Figure 1: Representation of the three studied configurations that take a $32 \times 32 \times 3$ image as an input and give a 10 dimensional output vector (for the 10 classes). [*conv.* for convolutional layer, *maxpool.* for maxpooling layer and *FC* for fully-connected layer]

3 The Training Procedure

3.1 The CrossEntropy Loss

The loss is defined by the result of the measurement error in the classification and we would like to minimize it by training our model. The standard loss for multi-class classsification is the cross entropy (Bishop, 1995, chap. 6) [2].

Let D be a database with a cardinal of N, we define the loss as following :

$$L(\theta) = -\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{C} q_{i,j}\ln(y_{i,j}) = \frac{1}{N}\sum_{i=1}^{N} L_i(\theta, \mathbf{q}_i, \mathbf{y}_i)$$

(4)

with $\{\mathbf{y}_i\}_{i=1}^N$ the output of our model (of parameter θ). The classes $\{c_i\}_{i=1}^N$ related to the images $\{\mathbf{X}_i\}_{i=1}^N$ are encoded by vectors $\{\mathbf{q}_i\}_{i=1}^N$ defined by :

$$q_{i,j} = \begin{cases} 1 \text{ si } c_i = j \\ 0 \text{ sinon} \end{cases} \quad \forall j \in \{1, ..., C\}$$

By minimizing L, we try to obtain, for each sample $i \in \{1, ..., N\}$ of the dataset, an output vector $\mathbf{y}_i$ such as its component $j = c_i$ has the higher value and the other ones have really small values.

3.2 Training algorithm

In a supervised context, the goal of the model training is to modify the parameters θ of the model in order to reach a loss minimum. The training procedure is an iterative process and because the loss could have numerous local minima, we use the loss gradient to modify the weights of the model in the direction of a local minimum.

In a normal situation and according to [1], I would have chosen the stochastic gradient descent (SGD) that modify the parameters for each training sample i, one by one. For the iteration $l + 1$, the weights update is defined as following :

$$\theta(l+1) = \theta(l) + \Delta\theta(l) \tag{5}$$

$$\Delta\theta(l) = -\mu\nabla_\theta \left[L_i(\theta, \mathbf{q}_i, \mathbf{y}_i)\right]_{\theta=\theta(l)} \tag{6}$$

with μ the learning rate that sets the importance of the update in the θ space, for the iteration l and in the direction of a minimum of L_i. $\nabla_\theta L_i$ is the gradient of L_i from the θ space that is computed thanks to the partial derivatives of L_i from each parameter. To avoid a noisy gradient, a batch of samples is generally used to compute the weights update from a mean gradient.

However, a SGD algorithm could take time to converge into a local minimum and because I do not have enough time to train my networks with this type of training algorithm (my training procedure is quite slow because I do not have GPU on my own computer), I decided to use the Adam one that converges faster [3]. As I understand the Adam algorithm, the principle is still the same but we compute, at each iteration, the optimal learning rate such as the weights update will have a big importance if we are in the right direction of the minimum and a small importance if we are not in this direction.

According to [1], we use 0.001 for the learning rate initialization (initialization because Adam will modify this value at each iteration) and a weight decay of 5.10^{-4} that penalizes the model weights with a L2 penalty resulting on smaller values of the weights. The batch size is set to 100 training samples during the training procedure.

Finally, it is necessary to fix a end of the training procedure. Normally, I will implement a early stopping procedure according to a validation dataset that allows us to avoid the overfitting issues. For that, the database is divided into a training set, a validation set and a test set. The overfitting happens when the model learns by heart the samples of the training set. In this case, the model becomes less generalizable because it learns also the noise of the training samples that are not useful for the classification problem. The validation set is used to evaluate the model, at each iteration of the training procedure, by computing the loss for those samples. If the validation loss stops to decrease, that means the model stops to learn useful information from the training set : it simply becomes more effective for the training samples but less effective for the validation samples. To avoid this overfitting situation, the training procedure is stopped just before. However, when I trained my models, I noticed that the validation loss and the training loss are really similar. That's why I do not use a early stopping method and simply stop my training procedure after 50 iterations.

4 Discussion

First, I studied three different CNN models, whose the configurations are previously described, in order to select the more relevant architecture for our problem and our dataset. To evaluate the performances of a deep neural network, it is really important to do multiple training procedures. Indeed, the performances could depend on the initialization of the weights which is random in our case and consequently, doing statistics is necessary to have the performances of those models regardless the initialisation of the model's parameters. Unfortunately, I have not been able to do a statistical study with the three networks trained on the entire training dataset.

Considering my hardware limitations and the requirement of statistical performances, I use a small trick which is doing the statistical study with models trained with a subset of the training database. For that, we consider a subset of 5000 training samples and a subset of 1000 test samples, that are randomly selected within the respective entire training and test sets. Because of the random selection procedure of the samples, it is indeed important to doing statistics. Unfortunately, because of the large number of my models' parameters and the fact that I worked without any GPUs, I was able to doing statistics with only 5 different training launches for each configuration. Depending of the configuration, it took me between 4 and 6 hours to train one model. I have to be aware that my conclusion could be biased because of the small statistics and the fact that we do not use the entire database available for us. For this study, the training procedure is stopped after 100 iterations because 50 iterations were not enough to reach the loss minimum with a subset of the database. The statistical study is described in the Table 1

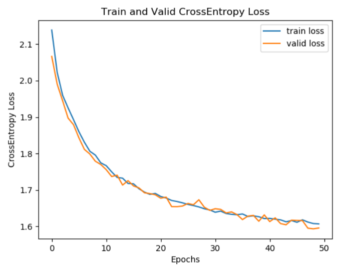
	Top-1 Accuracy [%]			Top-5 Accuracy [%]			Number of parameters
	Min	Mean	Max	Min	Mean	Max	
CNN A	54.9	59.3	62.6	93.9	94.3	95.0	591274
CNN B	51.1	52.9	54.9	90.1	91.1	92.5	550570
CNN C	33.3			76.9			1304746

Table 1: Statistical study over 5 training launches for each configuration (CNN A, CNN B and CNN C)

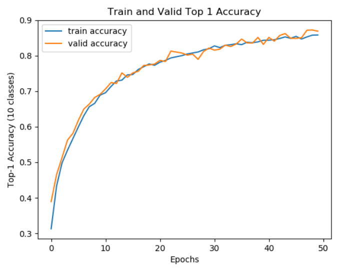
The CNN C is not able to converge properly with only 100 iterations and consequently I did only one try for this configuration. It could be explained by the large number of parameters (1304746) comparing to the size of the training set (5000) but also because, at the end of the stack of the *conv. units*, the output is a set of 256 feature images whose the size is 2×2. The feature image is feed inside the fully-connected part of the network and consequently transfers the information to the classification part of the model. Regardless the number of parameters, it seems that the CNN C reduce too much the size of the input images resulting to features with too poor information inside.

In the same way, we notice that the CNN B performs worth than the CNN A. For the CNN B, the output of the convolution part has a size of 4×4 (128) whereas the CNN A gives 8×8 (64) feature images to the full-connected part. It looks like it is easier to converge if we keep feature images with a higher size. Consequently, I keep the CNN A as the optimal configuration for the CIFAR-10 classification problem. Of course, those results is highly related to my hardware configuration and limitations as I explained before. But I will always prefer a smaller network that gives us pretty good results that a huge one that could not converge or overfit.

Finally, by training the CNN A with 50000 training images, we get 72.1 % of *Top-1 Accuracy* and 97.7 % of *Top-5 Accuracy* on the test set (10000 images). During the training procedure, we divide the training set into two subsets with a fraction of 0.3 : the validation dataset (15000) and the training dataset (35000). As we explained before, the validation samples are not used to update the weights of the model but only used to evaluate the model at each iteration. We obtain the evolution of the loss and the accuracy for both subsets.



(a) Cross Entropy Loss



(b) Top-1 Accuracy

Figure 2: Evolution of the loss and the accuracy over the training iterations for both validation and training datasets

We can note that both progress in the same way until the end of the training procedure : we do not have an overfitting situation. The performances on the training dataset are similar to those on the validation dataset, which suggests that the test performances will be closed as well. This trained model is provided inside the GitHub repository.

5 Conclusion

We implemented three different CNN configurations by adding different number of *conv. units* and we compared those models thanks to the CIFAR-10 dataset which provides 32×32 RGB images belonging to 10 different classes. We described mathematically the multi class classification problem and we presented the training procedure that is used to build CNNs. We also detailled the differents layers that are used to build our CNNs. In regards my hardware limitations, we concluded that the less depth configuration fits better with the low resolution images of the dataset available for us. It is important to take into account the fact that my hardware configuration doesn't allow us to study in a proper way those different architecture. However, we succeed to train a model that gives pretty good accuracy of classification on the test dataset. It will be really interesting to implement the ResNet CNNs that provide a nice trick to avoid overfitting and get better results according to [4].

References

[1] A. Z. Karen SimonYan, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556v6*, 2014.

[2] C. M. Bishop *et al.*, *Neural networks for pattern recognition.* Oxford university press, 1995.

[3] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[4] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.