Understanding Convolutional Neural Networks

Julià Camps JULIA.CAMPS.SEREIX@EST.FIB.UPC.EDU

Daniel Mora DANIEL.MORA.CHECA@GMAIL.COM

Abstract

This project is a coverage of the insight of how Convolutional Neural Networks (CNN) works in image classification problems and a practical use of this technology on a well-known dataset. We summarize the most important elements of a CNN architecture and apply this information into the CIFAR-10 dataset achieving successful results around the 65 % of accuracy.

Keywords: cifar10, convolutional neural networks

1. Problem statement and goals

This project is motivated by the thirst of knowledge of our team about deep learning techniques, specifically related to image classification. Deep Learning architectures have been used in recent years for solving multiple image classification problems for big amounts of data and, among them, Convolutional Neural Networks (CNN) have shown a great reputation.

We have reviewed some of the most relevant literature related to CNN and we have applied this knowledge in a practical scenario using the well-known CIFAR-10 dataset from Alex Krizhevsky and Hinton.

In 2 we summarize the related literature we have reviewed, in 3 we show a summary of the main concepts in CNN, 4 shows the architecture we have designed for our tests, 5 present the tests performed by our team, 6 comment the strengths and weaknesses of our project and 7 show the conclusions of our work.

1.1. Changes from the project's proposal

- The overwhelming amount of information related to Deep Neural Networks and the different approaches available made the project infeasible. We decided to focus on a specific type of networks within the so-called Deep Learning: Convolutional Neural Networks.
- The initial plan was to use the Google's library Tensorflow. However, the learning curve of the tool was very steep and documentation scarce. We decided to move to Lasagne library.
- Our initial thoughts were to use the Caltech-101 dataset from cal. This dataset was our first option but, as we have seen during our research stage, it shows a very low

performance using the selected technique, as pointed out in the work from (Zeiler and Fergus, 2013). This fact it is due to the 101 different classes contained in Caltech-101 dataset and some of them having very few instances (sometimes less than 40).

2. Related work

The idea behind Convolutional Neural networks was already present in the 60's on perceptrons connecting to local receptive fields of the input, visually inspired by the cat's visual system (see LeCun and Bengio, 1998). Fukushima was one of the first authors to introduce the concept of convolutional nets. But the expansion of these methods has has taken place in the last 10 years.

The evolution of the computing power of the computers and the increase of the data available had lead to deep architectures, as CNN, outperform humans in designing computer vision systems.

One of the most influential work in CNN is the form Krizhevsky et al. improving the state of the art on the ImageNet Challenge 2012. They defined a winning architecture that has been proved to work on multiple scenarios, as show in the paper from (Zeiler and Fergus, 2013), that also improves the previous results. Other works have also modified the main CNN architecture, such as the work from Le et al. (2010), than breaks the sharing weights schema to build more invariant features in the convolutional layers.

Many works reviewed have focused on the understanding of the CNN networks as the work from Nielsen (2015). Other literature analysed showed suggestions about useful configurations and practises for training this type of networks (see Jarrett et al.).

3. The CI methods

Feed forward neural networks have been used for multiple problems including image classification, since they can approximate any linear function. However, in order to learn possible variations in the data (e.g. translations) weights must be replicated across units and big amounts of data are needed to avoid overfitting. Images are usually flattened and fed into these networks and the order of the pixels in the feature vector is not relevant. However, this fact contradicts the nature of images, that have a very strong 2D structure.

Convolutional Neural Networks (CNN) are neural networks that have one or more convolutional layers followed by one or more fully connected layers. The purpose of the convolutional layers are to learn useful 2D local features without the expenses of learning a handful of parameters due to weight sharing of the neurons in the feature maps, achieving a faster training.

The following subsections detail some of the most important elements in the CNN framework.

3.1. Convolutional layers

The nature of the convolutional layers is possibly the most important characteristic of the CNN framework. They provide a supervised way to learn hidden patterns of the input images using shared weights, reducing the number of parameters to learn with respect to the feed forward fully connected networks.

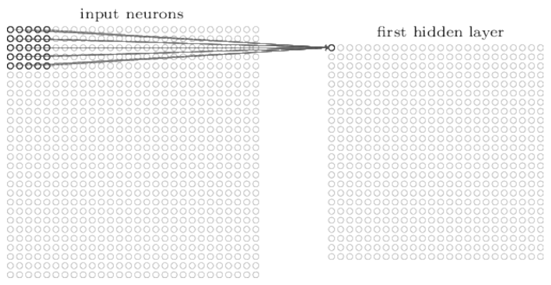

Figure 1: Example of a receptive field of the input layer connected to a neuron of a convolutional layer in 2 dimensions. Source: Nielsen (2015)

A convolutional layer takes an input I of size $d \times n \times n$. This input is convolved by using a squared *mask* of size z in a manner that each neuron in the convolutional layer only covers a region of I that is called *receptive field* (1). This receptive field is defined by the area of size $d \ \times z \ \times \ z$ starting at a certain point, where d are the number of channels of the input. The result of connecting each area of interest in I to a specific neuron simulates a sliding window convolution.

To achieve that a set of convolutions represent a uniform pattern in an image we must ensure that all receptive layers act in the same way. This is obtained by weight and bias sharing. Therefore, neurons that conform a feature share the same set of weights across all channels. A set of neurons that represent an image feature across several channels is called a *feature map* or a *kernel*. The number of neurons in a kernel k_i is:

$$neurons(k_i) = (n - z + 1)^2 \tag{1}$$

And has a $d \ \times z \ \times \ z$ weight matrix W_i, where d is the depth of the input layer. In each layer we compute several of these features where each one is activated on different edges, blobs or gradients of the input. The number of kernels in a convolutional layer determines the depth of the layer. Thus, the number of neurons in a layer l that contains k maps is:

$$neurons(l) = k \ \times \ (n - z + 1)^2 \tag{2}$$

And has a $k \ \times d \ \times z \ \times \ z$ weight matrix W^l.

The weight matrix of each map are the parameters of the convolution and are adjusted through training. Given a map k_i^l, from a layer l, its output y_i^l for a position (x, y) in I is:

$$y_{i,x,y}^l = \sigma(I_{x:x+z,y:y+z} * k_i^l) = \sigma(W_i^l \cdot I_{x+z,y+z,r}) + b) \tag{3}$$

Where operator $*$ is the convolution operator and $\cdot$ is the dot product between the two 3D matrices. The function σ represents the non-linearity applied to the convolution. Note that I may be the input image or the result of another convolutional layer. The vector b has size k and the bias of a map is shared by all neurons in the map. A summary of this process is depicted in B.

In this work we assume that images in a CNN are squared and so are filters, but it is not a restriction of the framework. We also assume that the size of the output is smaller than its input because of the convolution operation itself removing the outer borders. To preserve the input size we can use padding techniques that fill the removed pixels (usually with zeros).

It may also be the case than we want to reduce the overlapping between features from the input. The *stride* parameters determines the step between consecutive convolutions. Note that combining padding P and stride S modifies the number of neurons in a map h_i (see University):

$$neurons(h_i) = \frac{n - z + 2P}{S} + 1 \tag{4}$$

For simplicity we have assumed that $padding = 0$ and $stride = 1$ throughout this work.

3.1.1. Non-linearity

As we have seen before 3, after a convolutional layer usually a non-linear function is applied. Though sigmoid or hyperbolic tangent functions could be used, Krizhevsky et al. shows how using Rectified Linear Units (ReLUs)(see Nair and Hinton, 2010) have proven to provide a faster convergence in image classification problems. ReLU function f is defined as:

$$f(x) = max(0, x) \tag{5}$$

Using ReLU units has also been pointed as a key factor in convolutional neural networks in (see Jarrett et al.) and it is widely applied in this type of scenarios.

3.2. Local normalization layers

Another aspect that has proven to be effective for convolutional neural networks is local normalization. This technique is inspired by computational neuroscience experiments (see Jarrett et al.) and consists make the neighboring positions of a matrix to compete against each other. The most widely used normalisation approaches are:

- Normalization using neighboring pixels in the map. Given value $v_{k,x,y}$ (map k, position (x,y)) from the convoluted input of size S to normalize:

$$v'_{k,x,y} = \frac{v_{k,x,y}}{(t + \alpha \sum_{x'=max(0,x-\lfloor n/2 \rfloor)}^{min(S,x-\lfloor n/2 \rfloor+n)} \sum_{y'=max(0,y-\lfloor n/2 \rfloor)}^{min(S,y-\lfloor n/2 \rfloor+n)} v^2_{k,x',y'})^{\beta}} \tag{6}$$

  Where t, α and β are parameters in $\mathbb{R}$.

- Normalization across neighboring feature maps. Given value $v_{k,x,y}$ (map k, position (x,y)) from the convoluted input of size S to normalize:

$$v'_{k,x,y} = \frac{v'_{k,x,y}}{(t + \alpha \sum_{j=max(0,i-\lfloor n/2 \rfloor)}^{min(K,i-\lfloor n/2 \rfloor+n)} v^2_{j,x,y})^{\beta}} \tag{7}$$

Where K is the depth of the layer (number of maps) and α and β are parameters in $\mathbb{R}$.

- Hybrid approaches that take into account considering both neighboring pixels in the input and nearby maps.
- Derivations of the ideas proposed also exist.

3.3. Pooling layers

Typically, in order to obtain higher-level representations of the features and to achieve translation invariance, some sort of pooling is performed after a convolution. Pooling layers summarise the content of contiguous areas of size $p \times p$ along the depth of the input by computing a specific function on them. The depth of the input is preserved. Usually *mean*, *average* and *max* functions are used. The typical size used in pooling layers is $p = 2$, that downsamples the input maps by a half. Pooling is also helpful for reducing the dimensionality of the inputs.

3.4. Fully connected layers

Once normalised features are computed and pooled a model must be created in order to separate the different classes. This step is usually performed by feeding fully connected layers at the end of the convolutional set of layers.

3.5. Softmax layer

Image classification are usually multiclass containing multiple unique labels. One of the main techniques to address multiclass problems in neural networks is using a *softmax* activation function in the last layer. Considering a network with L layers, the output of the $i-th$ neuron of the last layer is defined as:

$$a_i = \frac{e^{z_i^L}}{\sum\limits_j e^{z_j^L}}, \quad z_i^L = \sum_k w_{ik}^L a_k^{L-1} + b_i^L \tag{8}$$

Where value a_k^{L-1} is the output of the $k-th$ neuron from the previous layer. This function has two main characteristics:

- It is always positive.
- The set of outputs always sum to one.

From these two statements we can assert that the set of outputs of the softmax layer represent a probability distribution. The output of a neuron j indicates the estimation of the inputs being from class j and an increase on the probability for one class implies reducing the probability of being from the other classes. Therefore, we will have as many neurons in this layer as possible classes in the problem.

3.6. Training CNN

3.6.1. Stochastic gradient descent

The optimization method selected to update the weights of our CNN is the Stochastic Gradient Descent (SGD). This procedure does not need the whole data set to update the weights of the network at each epoch but just a small set of data called batch. The updates in SGD are determined by a subset of data x_i and targets y_i:

$$w \leftarrow w - \alpha \bigtriangledown E(w; x_i, y_i) \tag{9}$$

This also makes the algorithm being dependent on the order it receives data. This is why data is usually shuffled before training.

In order speed up convergence a variant of SGD with momentum is usually used. It is useful in situations where the local optimum is in a valley and this valley is also a descending path along which the learning algorithm will *walk* very slowly. It modifies the previous equation with:

$$\begin{aligned} v = v \cdot m - \alpha \bigtriangledown E(w; x_i, y_i) \\ w \leftarrow w - v \end{aligned} \tag{10}$$

The value v is called the velocity and m is the momentum. This is more detailed in the work from Stanford.

3.6.2. The backpropagation algorithm

The error at each layer needed to approximate the corresponding derivatives in the CNNs can be computed using the classic backpropagation algorithm. However, specific considerations have to be made in order to define how this error is propagated through the convolutional, pooling and local normalisation layers. Due to the number of pages limitation this part has been skipped. Details can be found in the presentation from Kuwajima.

4. Architecture proposed

There is a huge amount of parameters that can be tuned in a CNN network and is unfeasible to parametrize all of them. In order to reduce the degrees freedom of our tests we decided to fix several of them according to our trial-error experience and to the literature reviewed.

Literature shows that an increase in the number of convolutional layers is translated into an accuracy gain (see Zeiler and Fergus, 2013). For technical reasons we will further detail, we set a maximum number of layers of these type. Jarrett et al. shows that a second convolutional layer has a big impact on the systems performance compared to a single one.

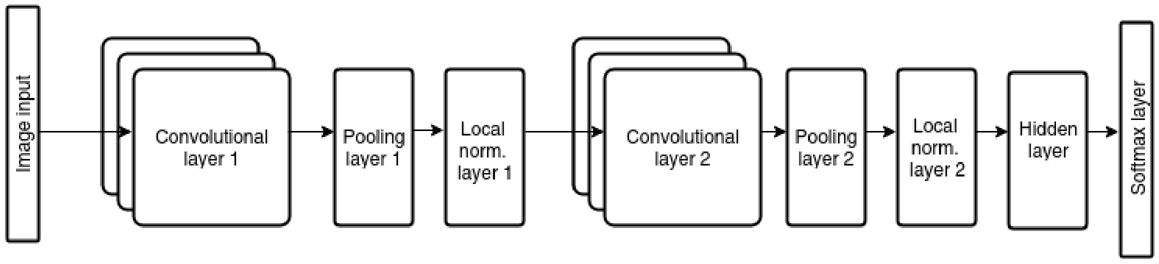

Figure 2: Basic architecture proposed for the CNN.

After each convolutional layer a pooling and a local normalization (across channels, using the default values from Lasagne) layer have been set. The filter sizes are 5 for both layers, stride has been set to 1 and *padding* has been disabled. Pooling layers use *max* functions.

We have set a single hidden layer with 256 units followed by a final softmax layer. Both layers have a dropout of 0.5. Dropout randomly deactivates (i.e. sets to 0) neurons with a certain probability in the layer to avoid overfitting (this mechanism is disabled during testing and evaluation).

All weights are initialized randomly following a Gaussian distribution centered at 0 and with standard deviation of 0.001 (following suggestions in work from Kuwajima) and all activation functions are ReLUs. The updates of the parameters is performed using Stochastic Gradient Descent with momentum of 0.9, learning rate of 0.01 and the loss function adopted is the cross entropy function.

The size of the batches for training, validation and testing phases has been set to 128.

A summary of the base architecture for our results is depicted in 2.

After the first convolutional layer the input is reshaped from 32×32 to 28×28 and the first pooling layer returns a 14×14 image. After the second convolution we get a 10×10 representation that is translated into a 5×5, that is the size (converted into a 25×25 vector that is fed into the hidden layer.

5. Results and Discussion

The dataset chosen for our practical scenario is a common challenge in computer vision, the CIFAR-10 classification problem. CIFAR-10 is a dataset that subsets a 80 tiny million images and consists of $60,000$ color images of 32×32 pixels that belong to one out of 10 categories: airplane, automobile, bird, cat, deer, dog, frog, horse, ship or truck. Some examples are displayed in A.

It is a good example to use with Convolutional Neural Networks because there are enough instances for each class in order to properly learn and images are heterogeneous across the image surface. Note that the CNN methodology may fail in scenarios where the relevant content in the image is limited to a specific area, such as face recognition datasets, and not uniformly distributed across the image. For the ease of the experiments we have taken a subset from the CIFAR-10 dataset:

- Number of instances for training: $30,000$ (from 3 training batches).

- Number of instances for validation: $2,000$ (from 1 training batch)
- Number of instances for testing: $10,000$ (test batch)

Though many strategies have been applied to speed up training, the procedure can still be very slow. A maximum number of maximum epochs has been set to 100 and an early stop strategy has been designed to halt training if the validation accuracy does not increase at least a 1% from the fifth previous epoch.

We have designed two tests:

- The first tests relates to the impact of varying the number of kernel maps per convolutional layer.
- The second test is related to the treatment of overfitting in convolutional nets.

5.1. Important notes about the experiments

The experiments we have designed are intended to be computed using GPU for being computationally expensive. However, none of the members of the team could finally configure the Lasagne library to work with its GPU interface due to issues with the drivers. Thus, the number of experiments and its variety has been affected by this fact. Results from our experiments are *one-shot* experiments and therefore strong conclusions cannot be derived.

Experiment 1(5.3) was performed in computer D and Experiment 2(5.4) was performed in computer C.

5.2. Data preprocessing

CIFAR-10 images are images that contain intensity values in the interval $[0, 255]$ at each of their 3 channels. Data whitening is widely applied in computer vision works but has proven not to be useful in convolution architectures when applied to small images (see Krizhevsky, 2009). We normalised data into interval $[0, 1]$.

5.3. Experiment 1: dealing with kernel maps

This experiment evaluates the impact of the variation of the kernel sizes on the system's accuracy. 1 shows how the performance of each configuration and their complexity related to the number of parameters and neurons they have. Results show that increasing the number of kernels slightly increases the test accuracy in expenses of adding complexity to the model and slowing down the training process.

There are some interesting results regarding the distribution of the kernel maps along the layers. Note that 30_70 and 50_50 networks have the same number of kernels but the first one contains around a 25% more parameters to train and has a 0.5% accuracy above. Although we cannot extract strong conclusions, we appreciate than the size of the second convolutional, and therefore, the number of higher level features, may be more relevant than lower level ones. This idea is supported also from other configurations (e.g 40_40 and 20_60).

Label	Kernels layer 1	Kernels layer 2	Neurons	Parameters	Epochs	Time	Test accuracy (%)
20_20	20	20	22,008	780,796	40	5:11:20	62.68
20_50	20	50	25,008	1,563,826	29	11:56:00	64.26
20_60	20	60	25,288	1,824,836	31	13:24:30	66.86
20_80	20	80	28,008	2,346,856	28	14:45:39	65.82
30_30	30	30	30,848	1,050,066	31	11:07:03	61.80
30_70	30	70	34,848	2,104,106	33	13:19:42	66.51
40_40	40	40	39,688	1,324,336	36	11:09:02	65.60
50_50	50	50	48,528	1,603,606	35	11:42:10	66.06

Table 1: Results for configurations using different number of kernels

5.4. Experiment 2: overfitting and ease of training

To avoid overfitting in deep neural nets several mechanisms can be applied, from which we analysed the traditional L1-regularization, L2-regularization and dropout. Dropout has been very effective in many different domains in expense of computational time (see Srivastava et al., 2014).

We tested several configurations using the default values presented in 4 and using 50 kernels for each layer. Regularization terms have been used with a coefficient of $1e-4$. 5 show the how the validation and the training loss function evolves along the epochs. Those settings with dropout show being slower at the early epochs but are successful to avoid overfitting performing a slow but steady descent. Configurations without dropout are early stopped around the $10th$ epoch due to generalisation degradation. This experiment supports the effectivity of the dropout for avoiding overfitting seen in the literature also supported by figure 6. We also have seen that dropout by itself outperforms dropout with L2 regularization but we do not have strong evidence to extract any conclusion.

5.5. Overall quality

We have uploaded our results into the CIFAR-10 Kaggle competition using our model 30_30 with dropout. It is not our best configuration but it was chosen due to technical reasons. Our score has been of 0.35150. This disappointing result that shows that our solution underfits the data. Possible solutions would be to use more data and also add more convolutional layers. Note that the winner of the Kaggle contest used up to 5 convolutional layers and our hardware limitations only allowed us to use 2. F shows the confusion matrix for that configuration on our test set. We observe how most of classes perform reasonably well except from *bird* and *cat* that are far below from the average. Model shows some bias toward automobile class.

6. Extensions, strengths and weaknesses

6.1. Extensions

Due our computational limited resources we have not been able to explore further interesting configurations of the parameters that could probably lead us to better results. Some of the experiments than we were not able to perform were:

- Changing the batch size: currently it is small. It could significantly improve the accuracy of our experiments.
- Experiments with adaptive learning rate should be planned. We have identified that it is important to learn fast during the first epochs but this may also cause to degrade the performance by overfitting the model in latter ones. It may be useful to reduce the learning rate by then.
- Testing our CNN on other datasets and on the full CIFAR-10.
- Check how the increase of convolutional layers affects the performance.

6.2. Strengths

Deep learning and CNNs are trendy concepts but were totally unknown for us. We have given a clear and brief insight about the most important elements in the CNN networks, how they are trained and practical situations where they can be applied with success.

6.3. Weaknesses

During the development of this project we faced problems than forced us to reconsider some of the initial ideas than were considered initially. The two most important issues during this work have been:

- Cross-validation of our results has been not possible (would have taken weeks) and strong conclusions have not been made from our experiments though interesting results have been obtained.
- Lack of experience with the chosen technologies made us reconsider the libraries to use several times due to the complexity and the lack of documentation of some of the options encountered (e.g. Tensorflow, Theano).

7. Conclusions

After this work we have built a strong understanding about how the CNN work and given an overview of how they work and how they can be tuned. Related literature has been reviewed showing that a lot of work is being done with this networks but still there is a lot to explore. Our experiments shown that dropout is a good choice for avoiding overfitting in this kind of networks and that kernel sizes are not very relevant in this dataset, though higher level features are more effective (and increase the network complexity). A real use case has been coded using Lasagne for Python giving test accuracies around 65% and with little generalization that could be addressed using more data and deeper architectures.

References

Caltech 101 dataset. URL http://www.vision.caltech.edu/Image_Datasets/Caltech101. 1

Cifar-10 competition on kaggle. URL https://www.kaggle.com/c/cifar-10. 13

Vinod Nair Alex Krizhevsky and Geoffrey Hinton. Cifar-10. 1

Kunihiko Fukushima. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological Cybernetics*, 36 (4):193–202. ISSN 1432-0770. doi: 10.1007/BF00344251. URL http://dx.doi.org/10.1007/BF00344251. 2

Kevin Jarrett, Koray Kavukcuoglu, and Yann Lecun. What is the best multi-stage architecture for object recognition? 2, 4, 6

Alex Krizhevsky. Learning Multiple Layers of Features from Tiny Images. Master's thesis, 2009. URL http://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf. 8

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C.J.C. Burges, L. Bottou, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012. URL http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf. 2, 4

Hiroshi Kuwajima. Backpropagation in convolutional neural network, 2014. 6, 7

Quoc V. Le, Jiquan Ngiam, Zhenghao Chen, Daniel Chia, Pang Wei Koh, and Andrew Y. Ng. Tiled convolutional neural networks. In *In NIPS, in press*, 2010. 2

Yann LeCun and Yoshua Bengio. The handbook of brain theory and neural networks. chapter Convolutional Networks for Images, Speech, and Time Series, pages 255–258. MIT Press, Cambridge, MA, USA, 1998. ISBN 0-262-51102-9. URL http://dl.acm.org/citation.cfm?id=303568.303704. 2

Vinod Nair and Geoffrey E. Hinton. Rectified linear units improve restricted boltzmann machines. In Johannes Fürnkranz and Thorsten Joachims, editors, *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, pages 807–814. Omnipress, 2010. URL http://www.icml2010.org/papers/432.pdf. 4

Michael Nielsen. In *Neural Networks and deep learning*. Determination Press, 2015. 2, 3

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15:1929–1958, 2014. URL http://jmlr.org/papers/v15/srivastava14a.html. 9

University Stanford. Stochastic gradient descent. URL http://ufldl.stanford.edu/tutorial/supervised/OptimizationStochasticGradientDescent/. 6

Stanford University. Cs231n convolutional neural networks for visual recognition. URL http://cs231n.github.io/convolutional-networks/. 4

Matthew D. Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. *CoRR*, abs/1311.2901, 2013. URL http://arxiv.org/abs/1311.2901. 2, 6

Appendix A. CIFAR-10 examples

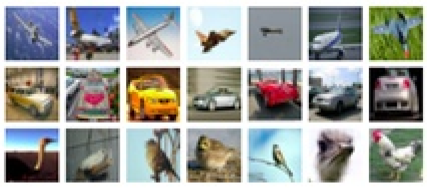

Figure 3: CIFAR-10 examples from three categories: plane(up), car (middle) and bird (bottom). Extracted from kag

Appendix B. Convolutional layers

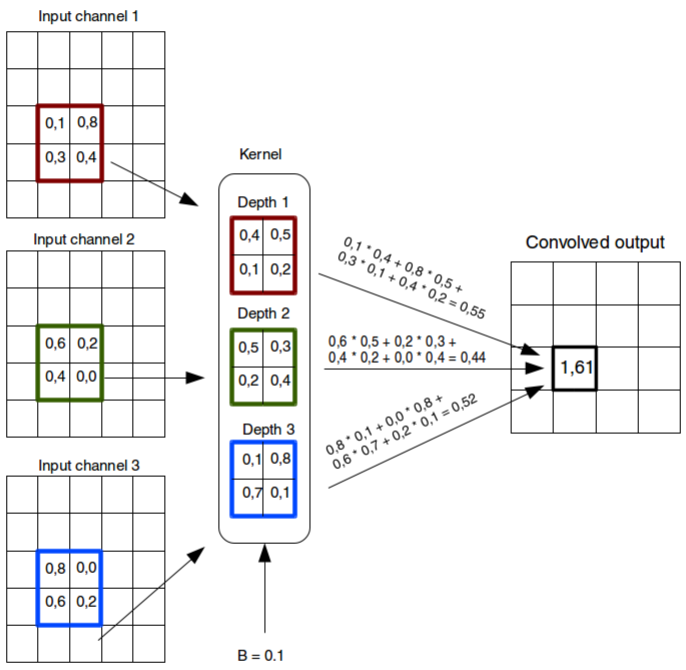

Figure 4: Visual description of 2×2 convolution on a region of the 3D input using a single map

Appendix C. Computer specs: Experiment 1

The first experiment has been tested on a 64-bit Linux computer with Intel i5 3210M with 4 CPUs at 2.50 Ghz and 3.8 GB of RAM.

Appendix D. Computer specs: Experiment 2

The second experiment has been performed on a 64-bit Linux system with a CPU Intel Core 2 Quad at 2.40 Ghz using 5 GB of RAM.

Appendix E. Experiment 2 results

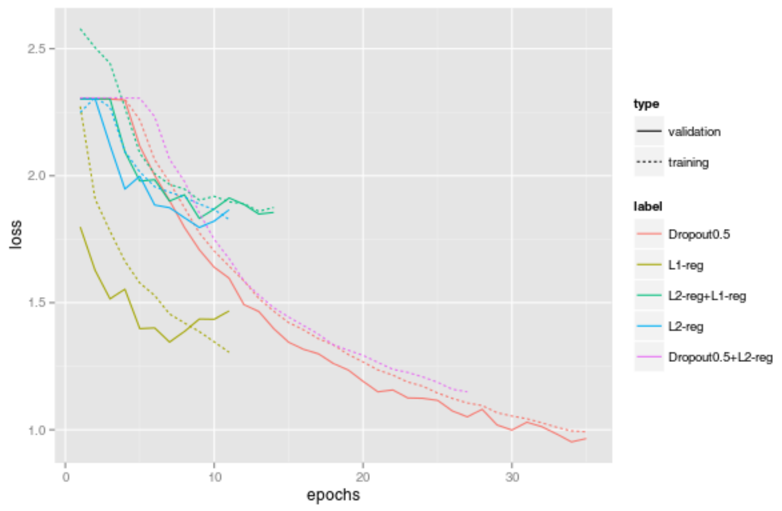

Figure 5: Evolution of the training and validation loss for different configurations

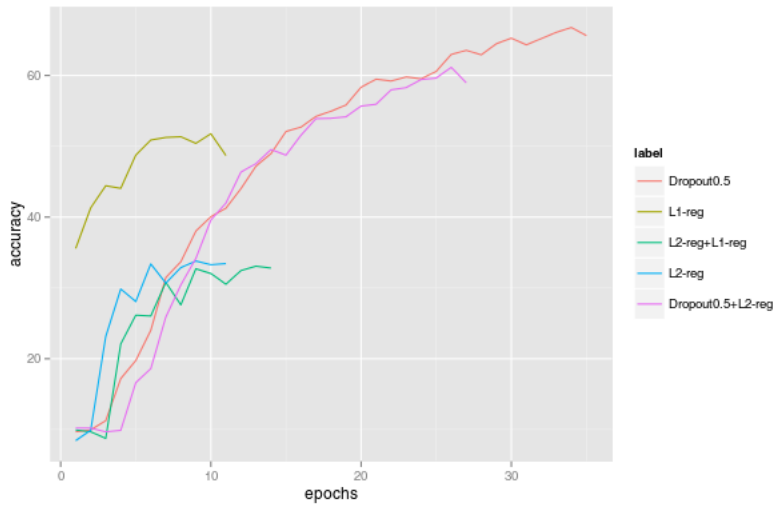

Figure 6: Evolution of the accuracies for different configurations (right)

Appendix F. Confusion matrix

Predicted / Truth	airp.	auto.	bird	cat	deer	dog	frog	horse	ship	truck
airplane	674	30	76	20	48	9	7	21	71	43
automobile	52	863	21	24	20	20	10	23	86	264
bird	46	3	415	65	90	62	40	24	6	7
cat	9	4	47	345	44	181	44	37	13	14
deer	13	2	94	65	504	46	38	64	8	3
dog	5	1	84	222	50	556	21	105	5	5
frog	16	21	129	191	134	84	793	25	13	24
horse	10	0	32	19	75	43	5	623	1	6
ship	151	36	40	34	21	10	11	16	781	38
truck	38	54	14	31	11	14	11	39	19	618

Table 2: Results for configurations using different number of kernels