SORBONNE UNIVERSITÉS

RDFIA Homework-2 :

Introduction to neural networks and convolutional networks for the image

Andrew Caunes
ZHAO Yunfei

January 13, 2022

This homework aims at learning basic theories of neural networks and programming and testing simple neural network as well as operations like backpropagation.

Contents

List of Figures

Chapter 1

Introduction to neural networks

1.1 Theoretical foundation

1.1.1 supervised dataset

1. ★ What are the train, val and test sets used for?

- The training set contains features and labels, it is given to the model to learn.
- The test set is given to the model only after the training, and is used to assess the true performance of the model. Testing on the training set would be biased because the model could have merely learned the features by heart and not be able to generalize. This set is completely isolated until the final evaluation of our model.
- The Validation set is used to assess the performance of the model during training. It is different from the training set so that the model can't overfit on this set. We tune the hyper-parameters by comparing the performance on this set.

2. What is the influence of the number of examples N ?

The number of examples needs to be enough to accurately represent the real distribution of the data. The more data, the better the model will be able to learn. In particular, neural networks require a very large amounts of examples.

1.1.2 Network Architecture (Forward)

3. Why is it important to add activation functions between linear transformations ?

There are two roles for the activation function:

- A model with only linear transformations would be equivalent to a linear classifier, no matter how many layers we add. The activation functions allows the model to be very flexible and learn a large range of functions.
- The activation also gives a range for the output to be contained in. For example, tanh make the output be in [-1,1] instead of all the real numbers with a mere linear transformation.

4. ★ What are the sizes nx, nh, ny in the figure 1? In practice, how are these sizes chosen?

- **nx** is the number of input feature in the data, thus it only depends on the data we are using.
- **nh** is the number of neurons in the model. It is chosen depending on the complexity we think the model should have. The more neurons, the more complex.
- **ny** is the size of the output of the network. For a classification task, we will generally choose it as the number of classes, so that each output feature indicates the confidence that the example belongs to one particular class.

5. What do the vectors $\hat{y}$ and y represent? What is the difference between these two quantities?

- Vector $\hat{y}$ represents the output of the model which will be used to classify the example. In our case, it represents the probabilities that the example belongs to each class, according to the learned model.
- Vector y represents the label for an example, that is the class it belongs to.

$\hat{y}$ is continuous in $[0, 1]$ while y is in $\{0, 1\}$.

6. Why use a SoftMax function as the output activation function?

The SoftMax activation function is allows us to obtain probabilities in $[0, 1]$ as an output. This is what we want for classification.

7. Write the mathematical equations allowing to perform the forward pass of the neural network, i.e. allowing to successively produce $\tilde{h}$, h, $\tilde{y}$ and $\hat{y}$ starting at x

$$\tilde{h} = W_h * x \tag{1.1}$$
$$h = tanh(\tilde{h}) \tag{1.2}$$

$$\tilde{y} = W_y * h \tag{1.3}$$
$$\hat{y} = SoftMax(\tilde{y}) \tag{1.4}$$

8. During training, we try to minimize the loss function. For cross entropy and squared error, how must the $\hat{y}_i$ vary to decrease the global loss function L?

- For cross entropy loss, given example (x_i, y_i), let j be the true label. y_i is zero everywhere except for its j-th feature, therefore the loss is decreasing when $\hat{y}_i$'s j-th feature increases to infinity.
- For squared error loss, the loss decreases when the $\hat{y}_i$ tends to y_i, that is when $(\hat{y}_i - y_i) \to 0$

9. How are these functions better suited to classification or regression tasks?

- Cross entropy loss is well suited for classification since it puts an heavy penalty on misclassified examples and zero loss on correctly classified examples. It is well suited for predictions which are probabilities in $[0, 1]$
- Squared error is well suited for regression as it takes predictions in any set of real numbers and gives a measure of the distance between the true value and the prediction.

1.1.3 Optimization algorithm

10. What seem to be the advantages and disadvantages of the various variants of gradient descent between the classic, mini-batch stochastic and online stochastic versions? Which one seems the most reasonable to use in the general case?

Batch gradient descent (or simply gradient descent) and Stochastic gradient descent are the two extreme versions of mini-batch gradient descent, when the size of the mini-batch tends to either N (Batch Gradient Descent) or 1 (Stochastic Gradient Descent).
A smaller mini-batch allows for :

- Faster updates since computed on less examples
- More randomness in the updates which can help explore and not get stuck in local optima.

While a large mini-batch will help with :

- Better directions for the update since computed on more data
- More potential for parallel computing

11. ★ What is the influence of the learning rate η on learning?

η is the most important hyperparameter for learning. The larger it is, the faster the descent will be, but if it is too large, the algorithm won't converge because the optimum will be missed since the steps will be too big.

12. ★ Compare the complexity (depending on the number of layers in the network) of calculating the gradients of the loss with respect to the parameters, using the naive approach and the backprop algorithm.

Let n_j be the number of parameter in layer j, the width $w = max_j(n_j)$, L the number of layer and let c be the complexity for an elementary computation.

$$\begin{aligned} C(Naive) &= \sum_{i=1}^{L}\sum_{j=1}^{n_i}\prod_{k=i+1}^{L} n_k * d \\ &= O(L * w * w^(L) * d \\ &= O(L * w^{L+1}) * d \end{aligned}$$

$$\begin{aligned} C(BackPropagation) &= \sum_{i=1}^{L-1}\sum_{j=1}^{n_i} n_{i+1} + n_L \\ &= O(L * w^2) * d \end{aligned}$$

The naive method is much slower with regards to the time complexity, even though it's slightly lighter considering memory usage.

13. What criteria must the network architecture meet to allow such an optimization procedure ?

We require each transformation in the network to be differentiable in order to use the Chain rule.

14. The function SoftMax and the loss of cross-entropy are often used together and their gradient is very simple. Show that the loss can be simplified by:

$$\ell = -\sum_i y_i * \tilde{y}_i + log(\sum_i e^{\tilde{y}_i})$$

Let $\ell(y, y') = -\sum_i y'_i log(y_i)$ We have

$$\begin{aligned}\ell(SoftMax(\tilde{y}), y) &= -\sum_i y_i log(\frac{e^{\tilde{y}_i}}{\sum_j e^{\tilde{y}_j}}) \\ &= -\sum_i y_i\tilde{y}_i + \sum_i y_i log(\sum_i e^{\tilde{y}_i}) \\ &= -\sum_i y_i\tilde{y}_i + log(\sum_i e^{\tilde{y}_i})\end{aligned}$$

15. Write the gradient of the loss (cross-entropy) relative to the intermediate output $\tilde{y}$

We get

$$\Delta_{\tilde{y}}\ell = \begin{bmatrix} \frac{\partial \ell}{\partial \tilde{y}_1} \\ \vdots \\ \frac{\partial \ell}{\partial \tilde{y}_{n_y}} \end{bmatrix} = -y + SoftMax(\tilde{y}) = \hat{y} - y$$

16. Using the backpropagation, write the gradient of the loss with respect to the weights of the output layer $\Delta_{W_y}\ell$. Note that writing this gradient uses $\Delta_{\tilde{y}}\ell$. Do the same for $\Delta_{W_b}\ell$.

We have

$$\begin{aligned}\forall i, j, \frac{\partial \ell}{\partial W_{y,ij}} &= \frac{\partial \ell}{\partial \tilde{y}_i} * h_j \\ \forall i, \frac{\partial \ell}{\partial b_{y,i}} &= \frac{\partial \ell}{\partial \tilde{y}_i}\end{aligned}$$

Thus,

$$\begin{aligned}\Delta_{W_y}\ell &= \Delta_{\tilde{y}}\ell * h^\top \\ \Delta_{b_y}\ell &= \Delta_{\tilde{y}}\ell\end{aligned}$$

17. Compute other gradients : $\Delta_{\tilde{h}}\ell$, $\Delta_{W_h}\ell$, $\Delta_{b_h}\ell$.

We have

$$\begin{aligned}\Delta_h\ell &= (W_y)^\top \Delta_{\tilde{y}}\ell \\ tanh'(x) &= 1 - tanh(x)^2\end{aligned}$$

Thus

$$\Delta_{\tilde{h}}\ell = (W_y)^\top \Delta_{\tilde{y}}\ell * (1 - h^2)$$

where * represent the element-wise product.
Then we conclude

$$\begin{aligned}\Delta_{W_h}\ell &= \Delta_{\tilde{h}}\ell\, x^\top \\ \Delta_{b_h}\ell &= \Delta_{\tilde{h}}\ell\end{aligned}$$

Chapter 2

Convolutional networks for the image

We will study the classical convolutional networks layers and we will set up a first network trained on the standard CIFAR-10 dataset. Once the first network is tested, we will discuss different techniques to improve the network learning process : **normalization**, **data augmentation**, **variants of SGD**, **dropout** and **batch normalization**.

2.1 Introduction to convolutional networks

1. Considering a single convolution filter of padding **p**, stride **s** and kernel size **k**, for an input of size **x** × **y** × **z**
what will be the output size ?
How much weight is there to learn ?
How much weight would it have taken to learn if a fully-connected layer were to produce an output of the same size ?

The kernel size is (k × k × z) because it should have the same size of input image for the third dimension and as we have only one filter, the size of third dimension of the output is 1. We set h and w as the first two dimensions of the output. In the case we do not do the convolution operation if there is no enough space, we have:

$$w = \lfloor \frac{x+2p-k}{s} \rfloor + 1 \tag{2.1}$$

$$h = \lfloor \frac{y+2p-k}{s} \rfloor + 1 \tag{2.2}$$

So the final output is of size (h, w, 1).
The weights to learn are the weights of kernel, so weights to be learnt are:

$$k^2 z$$

If we want to produce an output of the same size, the number of weights is supposed to be

$$x \times y \times z \times w \times h$$

.

2. ★ What are the advantages of convolution over fully-connected layers ? What is its main limit ?

First of all, we need to train fewer parameters for a convolution layer that conduct the same output size as a fully connected layer. Secondly, A neuron in a convolutional layer, however, is only connected to "nearby" neurons from the preceding layer within the width of the convolutional kernel. In this way it can concentrate more on local information, for example, to do an object classification for a cat in the center of image, we may only look at the pixels no far from this cat, while the fully connected layer will be influenced by pixels from the whole image, pixels from background, etc.

The main limit of of convoluational layer is that the narrow receptive field limite the information we can get from one layer, to have larger receptive field, we need to build deeper CNN.

3. ★ Why do we use spatial pooling ?

Spatial pooling can reduce parameters we need to train, but more importantly, it helps to get invariant representations across lateral shift of image local information. Intuitively, for an image of human face, if we zoom it, it is still a human face. Spatial pooling in a hierarchical feature model makes each feature resilient to minor shifts in position, since multiple positions are pooled together. If this technique is applied repeatedly and hierarcically, then an object can be recognized despite substantial spatial distortions, making object recognition more robust and less fragile. [1]

4. ★ Suppose we try to compute the output of a classical convolutional network for an input image larger than the initially planned size (224 × 224 in the example). Can we (without modifying the image) use all or part of the layers of the network on this image ?

Yes, we can use the whole convolutional layers and change the size of the fully-connected layers.

5. Show that we can analyze fully-connected layers as particular convolutions.

For a fully-connected layer that has an input size of the total pixel number of an image $x \times y$ and an output size C, We can build a convolutional layer with convolution kernel of size $x \times y$ (the same size of the image) and C filters. These two have the same number of parameters and the same operation.

6. Suppose that we therefore replace fully-connected by their equivalent in convolutions, answer again the question 4. If we can calculate the output, what is its shape and interest ?

Yes, we can still use the whole neural network, and we just need to adjust the last convolution layer as the size of the output feature map.

7. We call the receptive field of a neuron the set of pixels of the image on which the output of this neuron depends. What are the sizes of the receptive fields of the neurons of the first and second convolutional layers ? Can you imagine what happens to the deeper layers ? How to interpret it ?

For the first layer, with kernel size k1, the receptive field size is $k1^2$. For the second layer, with second layer kernel size k2, the receptive field size is $k2^2 \times k1^2$. So when the layer is deeper, the receptive field size will increase very quick as

$$\prod_{i=1}^{n} k_i^2, with\ k_i\ the\ kernel\ size\ of\ ith\ layer$$

It will soon take the whole image as receptive field.

2.2 Training from scratch of the model

2.2.1 Network architecture

The network that we are going to implement has a style close to the AlexNet architecture of Krizhevsky et al. (2012) adapted to the CIFAR-10 base whose images are smaller. It will be composed of the following layers :

- conv1 : 32 convolutions 5 × 5, followed by ReLU
- pool1 : max-pooling 2 × 2
- conv2 : 64 convolutions 5 × 5, followed by ReLU
- pool2 : max-pooling 2 × 2
- conv3 : 64 convolutions 5 × 5, followed by ReLU
- pool3 : max-pooling 2 × 2
- fc4 : fully-connected, 1000 output neurons, followed by ReLU
- fc5 : fully-connected, 10 neurons output, followed by softmax

8. For convolutions, we want to keep the same spatial dimensions at the output as at the input. What padding and stride values are needed ?

We set **stride as 1**, because we may use a stride >1 with a very large padding, but this makes no sense. So by following 2.2 we deduce that

$$2p - k + 1 = 0$$

We note that the kernel size need to be odd, or we need to have different padding number for each side of the input.

9. For max poolings, we want to reduce the spatial dimensions by a factor of 2. What padding and stride values are needed ?

If we want to reduce the spatial dimension by a factor of 2, the **stride size is set to 2**, and we have

$$\begin{cases} k - 2p - 2 = 0 & \textit{if } x \text{ - } k \textit{ is even} \\ k - 2p - 1 = 0 & \textit{if } p\ x \text{ - } k \textit{ is odd} \end{cases}$$

10. ★ For each layer, indicate the output size and the number of weights to learn. Comment on this repartition.

We are following Q8 and Q9's principles. For convolution layer, we keep the same spatial dimensions and for pooling layer, we reduce the spatial dimensions by two. With input image from CIFAR-10 of size 32 × 32 × 3 and if we do not count the bias parameters of convolutional layer (it is equal to the number of filters) We have number of parameters to learn:

$$\begin{cases} Input\ channel \times Kernel\ parameters\ size \times Output\ channel \quad For\ convolutional\ layer \\ 0 \quad For\ pooling\ layers \\ inputs \times outputs + biases \quad For\ fully\text{-}connected\ layers \end{cases}$$

- conv1 : output size: 32 × 32 × 32, weights to learn: 32 × 5 × 5 × 32 = 25600
- pool1 : output size: 16 × 16 × 32, weights to learn: 0
- conv2 : output size: 16 × 16 × 64, weights to learn: 32 × 5 × 5 × 64= 51200
- pool2 : output size: 8 × 8 × 64, weights to learn: 0
- conv3 : output size: 8 × 8 × 64, weights to learn: 64 × 5 × 5 × 64 = 102400
- pool3 : output size: 4 × 4 × 64, weights to learn: 0
- fc4 : output size: 1000, weights to learn: 4 × 4 × 64 × 1000 = 1024000
- fc5 : output size:10, weights to learn: 1000 × 10 = 10000

We have a little bit more parameters for features extraction on convolutional layer than classification layer and more parameters with feature of higher dimension (deeper convolutional layer) which seems reasonable.

11. What is the total number of weights to learn ? Compare that to the number of examples.

The total number of weights to learn is 25600 + 51200 + 102400 + 1024000 + 10000 = 2034800 parameters. While CIFAR-10 consists of 60000 32x32 colour images in 10 classes, with 6000 images per class. There are 50000 training images and 10000 test images. So it is relatively small compare to the number of parameters to learn.

12. Compare the number of parameters to learn with that of the BoW and SVM approach.

This method has mucn more parameters than BoW and SVM approach. For the last we just need to create our Bow and to train the parameters in SVM. For example, we use SIFT method to encode 16 × 16 small image patch into 128 dimensions vector. We use step size of 8 to get all vectors from each image in train set, and we build our dictionary by kmeans methods to have for example 1001 words, which is 128128 parameters to learn. Then we transfer each image into a vector by BoW to a vector of size 1001, this conduct (1002 × number of class) for N svm classifier if we use one to N strategy. But the total number is still quite small than CNN, but CNN can be easily parallelized and trained end-to-end.

2.2.2 Network learning

13. Read and test the code provided.

The result is showed in figure 2.2.2

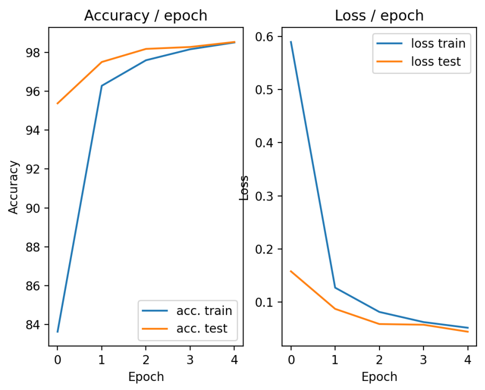

Figure 2.1: LeNet5 on MNIST epoch, epoch accuracy and loss

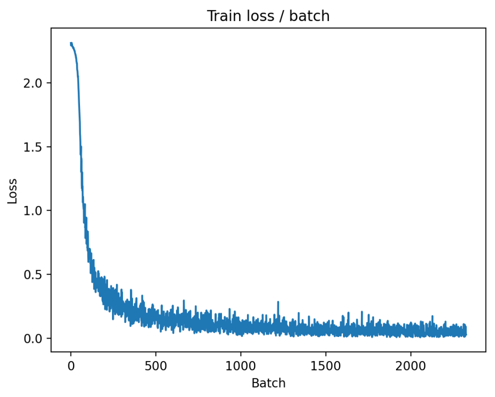

Figure 2.2: LeNet5 on MNIST train batch loss

14. ★ In the provided code, what is the major difference between the way to calculate loss and accuracy in train and in test (other than the the difference in data) ?

Except that they have different data set, the way to load the data set is different. For train, data is shuffled and for test data is loaded in order and the model mode is different for these two process. Further more, we update the model during train and during test we do not.

15. Modify the code to use the CIFAR-10 dataset and implement the architecture requested above. (the class is datasets.CIFAR10). Be careful to make enough epochs so that the model has finished converging.

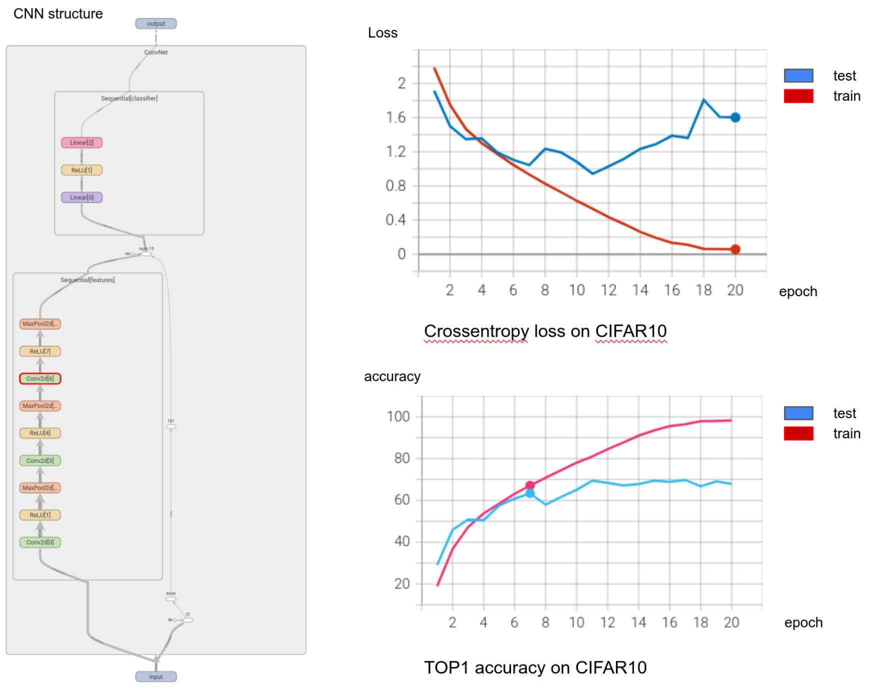

Figure 2.3: Experiment results on CIFAR10

16. ★ What are the effects of the learning rate and of the batch-size ?

Learning rate is the rate we use to do gradient descent for parameters, with larger learning rate, we will have bigger gradient step. And batch size is the size of samples we use for one gradient descent operation. With larger batch size, we tend to have a gradient generate by more samples and with potentially smaller variance than the gradient generated by only one sample.

17 and 18. ★ What is the error at the start of the first epoch, in train and test ? How can you interpret this ? Interpret the results. What's wrong ? What is this phenomenon ?

The error at the start of the first epoch is conducted by our network with random weights, so it can be relatively high while for the first epoch on test set, the model is already trained for one epoch on train, so there is a period that the result on test set is better than results on train set. And also due to the number of samples on train set is 5 times of which in test set, at the beginning, we can observe the **under-fitting** phenomena of our model.

2.3 Results improvements

2.3.1 Standardization of examples

19. Add normalisation on image channels. Describe your experimental results.

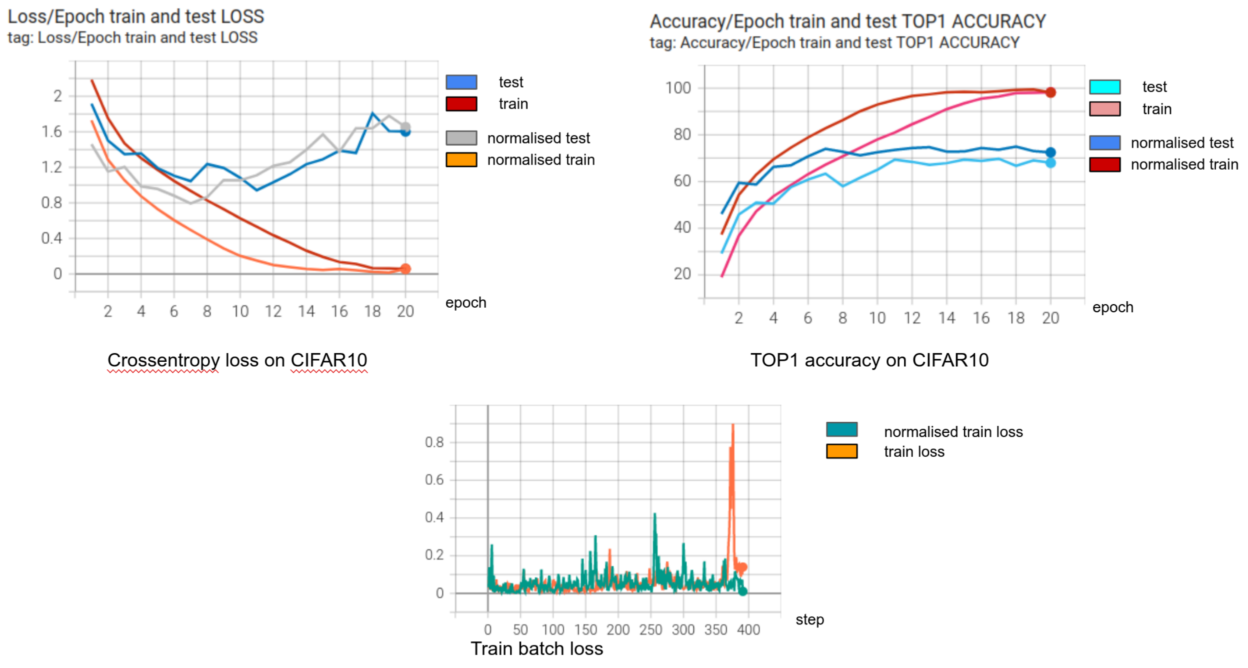

Figure 2.4: Compare of experiment results on CIFAR10 with and without normalisation on image channels

We can conclude that with channel normalisation, model converge quicker to a better result, and batch loss is relatively stable.

20. Why only calculate the average image on the training examples and normalize the validation examples with the same image ?

Because we suppose that the validation examples is here for testing our model, so we can not use any of its information for training, what we can do it that we suppose the samples in validation set and test set follow the same distribution as what we have in train set.

21. Bonus : There are other normalization schemes that can be more efficient like ZCA normalization. Try other methods, explain the differences and compare them to the one requested.

Need to do !!!!

2.3.2 Increase in the number of training examples by data increase

22. In train a random crop size 28 and a random horizontal symmetry and in test a centered crop size 28. Describe your experimental results and compare them to previous results.

From figure 2.5, we can conclude that by using data augmentation , we handle the over-fitting

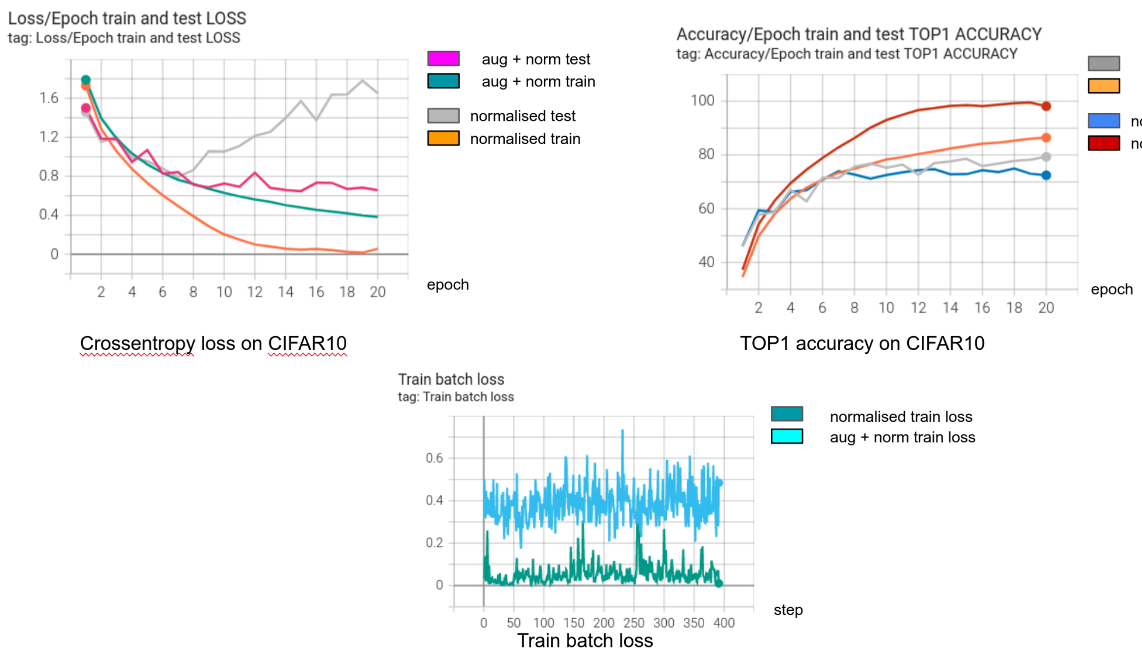

Figure 2.5: Compare of experiment results on CIFAR10 with and without data augmentation after normalisation on image channels

problem we observe better loss and accuracy on test data set.

23. Does this horizontal symmetry approach seems usable on all types of images ? In what cases can it be or not be ?

Horizontal symmetry approach is not usable when the signification of an image depends on the horizontal direction. For example, we want to our model learning to recognize time by images of dial and the numbers on dial is not clear, then horizontal symmetry approach may completely change the meaning of this image. Besides, if we want to do image description task, this operation can change the meaning of image.

24. What limits do you see in this type of data increase by transformation of the data set ?

Firstly, we found that with these processes, the time we use to load data is longer, but we can solve it by add more workers of CPU. Secondly, for some images, we are not sure that with data transformations, the underlying meaning of these images is not changed. And the training processes may not be very efficient as we already used information of images and when we transform them, the model need to learning all the same way.

25. Bonus : Other data augmentation methods are possible. Find out which ones and test some.

Yes, Such as rotation, vertical flip, zoom in and out, or combine several images as one image for object detection.

26. Modify the optimizer SGD to add momentum to it (see the documentation) using the standard value of 0.9. Add a learning rate scheduler and use an expoential decay with a coefficient of 0.95. Describe your experimental results and compare them to previous results, including learning stability.

With momentum by considering the former gradients and decreasing learning rate, we found

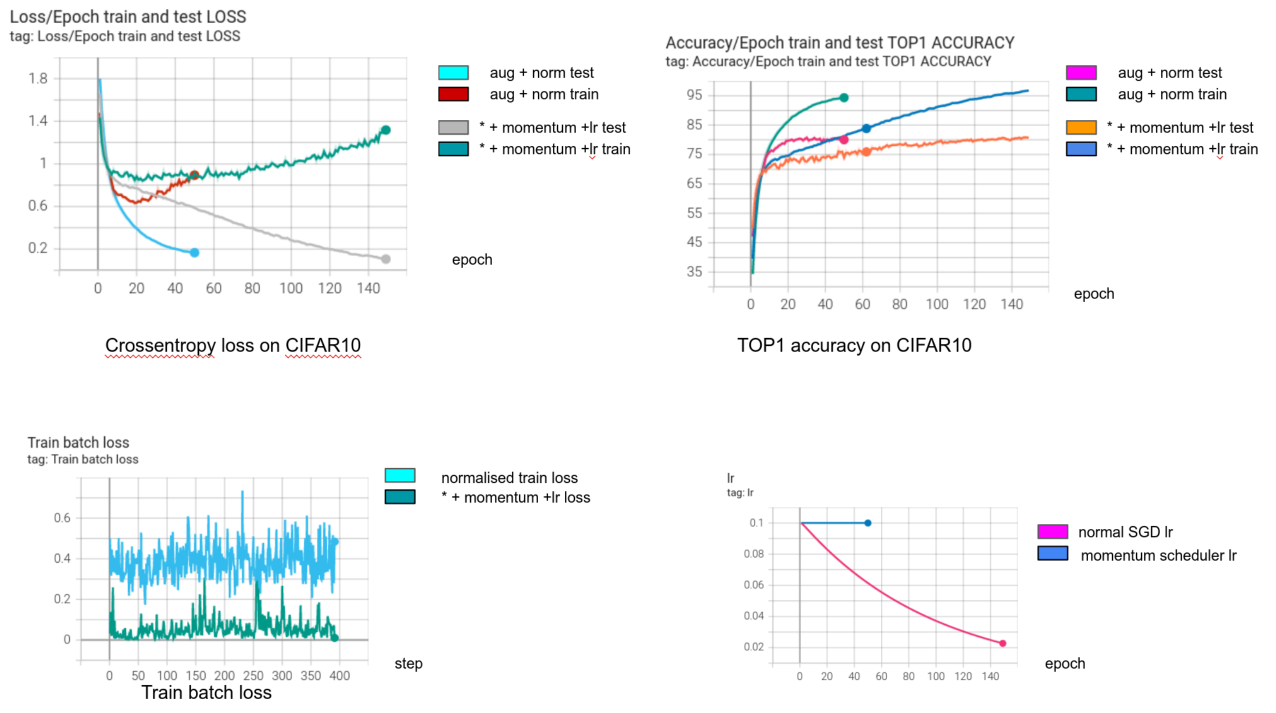

Figure 2.6: Compare of experiment results on CIFAR10 with and without momentum and learning rate scheduler

that learning process is surely more stable. But it takes more training epoch to become lightly better than normal SGD optimiser. We tried several hyper-parameters, include momentum and scheduler gamma. **This is a interesting issue to discuss**.

27. Why do these two methods improve learning ?

It supposes to achieve better results. Because with momentum, we shall consider former gradient generated by former samples so this lead to a gradient stabler and more reliable. While during training, when parameters are converging to optimum, it is better to take little step, with large step it may become divergent.

28. Bonus : Many other variants of SGD exist and many learning rate planning strategies exist. Which ones ? Test some of them.

For example, for optimiser there are Adma, RMSprop, etc. And for scheduler, there are LinearLR, LambdaLR, etc. There is a clear list on https://pytorch.org/docs/stable/optim.html. And we tested Adma as it is a famous optimiser which

2.3.3 Variants on the optimization algorithm

2.3.4 Regularization of the network by dropout

29. Add a dropout layer between fc4 and fc5. Describe your experimental results and compare them to previous results.

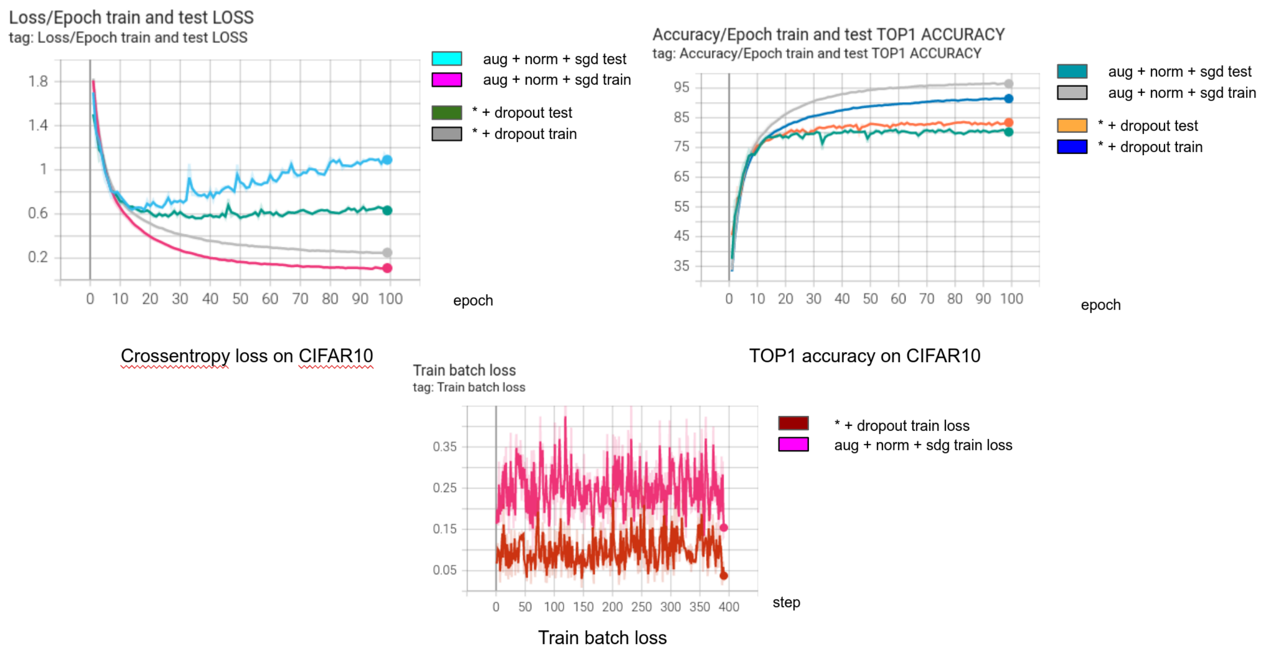

Figure 2.7: Compare of experiment results on CIFAR10 with and without dropout

We find out that with dropout, there are fewer over-fitting problems.

30. What is regularization in general?

Regularization is the process of adding information in order to solve an ill-posed problem or to prevent over-fitting. Regularization can be motivated as a technique to improve the generalizability of a learned model. (Wikipedia). We think that, in our case, the dropout is a kind of regularization in the number of parameters of our model.

31. Research and "discuss" possible interpretations of the effect of dropout on the behavior of a network using it ?

We can tune the hyper-parameter p to see the effect by evaluating the model use it on validation data data set. Furthermore, we can visualise the feature map after the dropout layers if it is applied to convoluational layers.

32. What is the influence of the hyper-parameter of this layer?

The hyper-parameter **p** is the probability that the output of a node is dropped out. So the expectation of final number of weights used during forward propagation is $N \times (1 - p)$. With higher **p**, the regularization is stronger.

33. What is the difference in behavior of the dropout layer between training and test ?

For during training, dropout layer with randomly set values of the output of the precedent layer to zero with a probability p. During test or validation, the dropout layer will work as identity function. To be noted that durign training, we need to divide the output of the dropout layer by (1 - p) to scale it as it has fewer elements and we want it to look like that it works as whole weights (this is implemented in Pytorch dropout function). And normally, we put dropout function after activation layer.

2.3.5 Use of batch normalization

34. Add a 2D batch normalization layer immediately after each convolution layer. Describe your experimental results and compare them to previous results.

Batch normalisation layer learns to re-normalize the outputs of the previous layer, allowing to stabilize the learning.

From figure 2.8, we can conclude that, batch normalisation surly help the optimisation process. With this technology we improved our result based on all other methods we used before. It achieves 85.53% accuracy on test set.

★ Implemented method. In this part, we tested all demanded methods and some of the bonus, except Bonus Q21. Thanks for this homework, we have a better understanding of deep learning and it's application in computer vision. It is a perfect announcement with clear explanations!!!

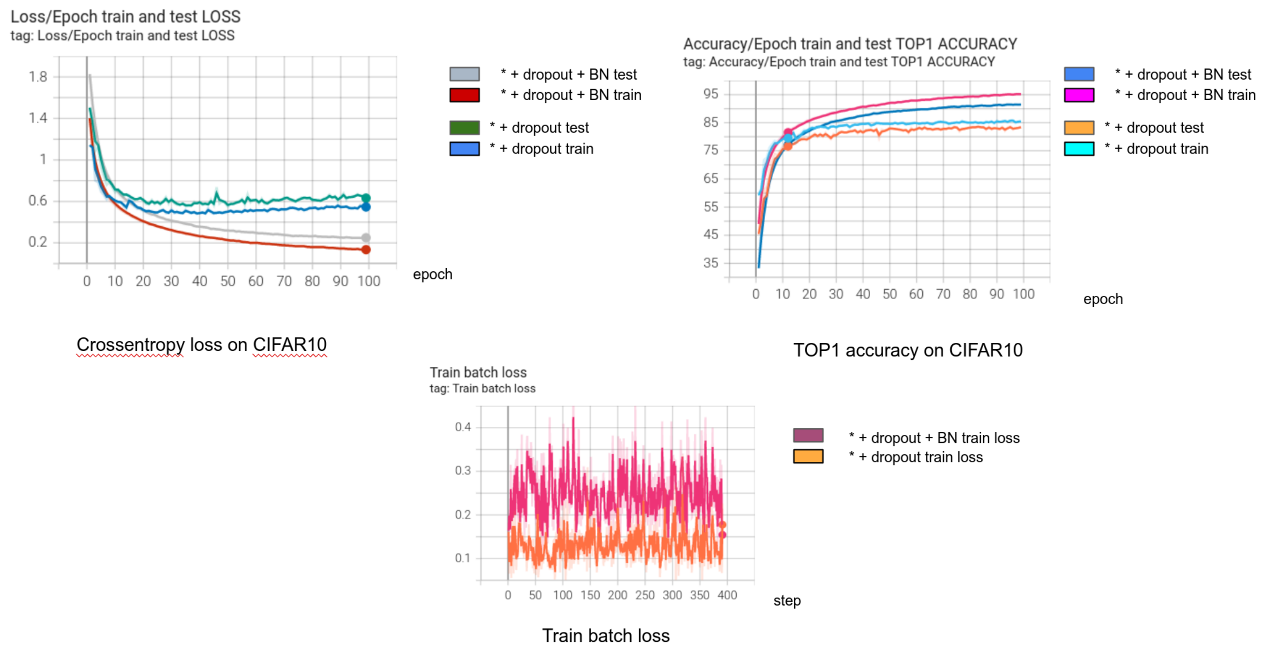

Figure 2.8: Compare of experiment results on CIFAR10 with and without BN, it has dropout on fully connected layer with SGD and image channel normalisation

Bibliography

[1] *What is spatial pooling in computer vision?* https://www.quora.com/What-is-spatial-pooling-in-computer-vision.