Assignment 2 : Selected Topics in Distributed Multimedia Systems

Baptiste CALLARD

August 12, 2022

Table des matières

I Introduction . . . 1
II Presentation of results . . . 1
II.1 Reference model . . . 1
II.2 type of the padding . . . 1
II.3 Use ADAM . . . 2
II.4 dropout . . . 2
II.5 batch normalisation . . . 2
II.6 kernel size . . . 3
II.7 Add convolutional layer . . . 3
II.8 Add fully connceted layer . . . 3
II.9 batch size . . . 4
II.10 number of neurons per layers . . . 4
II.11 Proposal of an optimised model . . . 5
III Conclusion . . . 5
IV APPENDIX . . . 6
IV.1 Images . . . 6
IV.2 Intermediate models structures . . . 6
IV.3 Best model . . . 6
IV.4 Best model without Batch . . . 7

I Introduction

The aim of this assignment is to follow up on the first assignment. It will also serve to apply the theoretical knowledge on a concrete example. This second assignment corresponds again to a supervised multi-class classification problem. This time we will use the CIFAR-10 [1] dataset. I have also chosen for this assignment to do it on Python.

II Presentation of results

I have chosen to study the impact of several parameters. For this, I chose to use a reference model and to return to it systematically to make my comparisons. Thus, I tested to vary only one parameter at a time to know what is its real impact. The parameters I chose to look at are : padding type, dropout, optimizer, batch normalisation, kernel size, add convolutional layer, add fully convoltional layer, batch size, number of neurons per layers. As a reference model, I chose a model similar to the one in the R file.

II.1 Reference model

Here is the reference model, it is a relatively simple model with a parameter number of 2,175,914. I used a batch = 64, epoch = 15 and 20,000 images for training and 2000 images for testing. I would use the same architecture and parameters in the following sections except the section where I will add layers etc... In this case I will specify the structure in appendix.

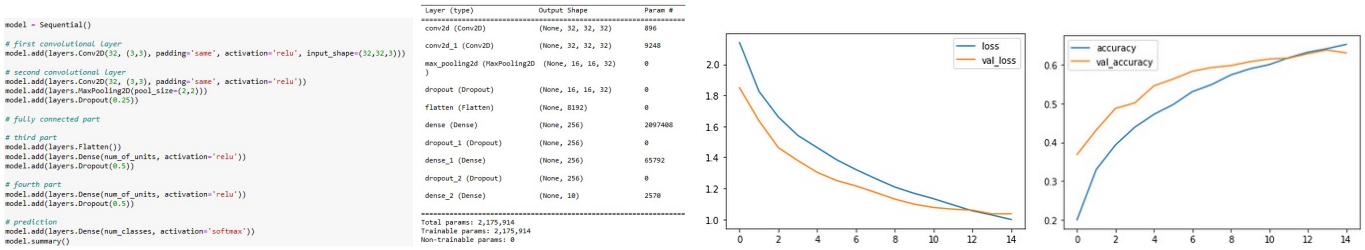


FIGURE 1 – Reference model and performances

We can see that after 15 epochs the model reaches around 65% for val_accuracy. In addition, we can see that the val_loss curve starts to stagnate and stats aloso to increase which suggests that the model is starting to overfit and therefore the model is not big enough, deep enough or does not have the right structure to learn properly. This model dropout as normalisation.

II.2 type of the padding

In this section I have replaced the padding same by the padding valid. The same padding allows to have the same dimension between the input and output data. For this, zeros are artificially added on the edges to be able to apply the kernel even to the nodes on the ends.

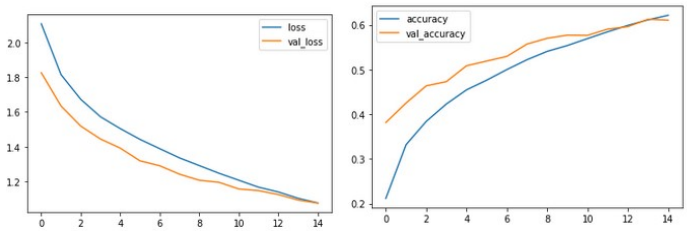


FIGURE 2 – model with padding = valid and performances

1. airplane, automobile, bird, cat, deer, dog, frog, horse, ship, truck

There is no significant change in our case. On the other hand, there are fewer weights to learn because there are only 1,684,394 parameters. So with "valid", the execution time is quicker than the other. Thus, "valid" seems to be a better option than "same".

II.3 Use ADAM

Instead of using RMSprop, I tried to use ADAM.

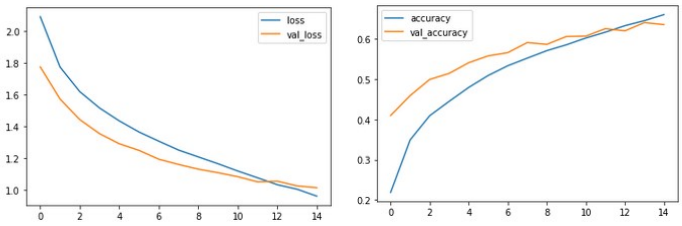


FIGURE 3 – model with the utilisation of Adam optimizer

Clearly the performance is equivalent. I also looked at the execution time. For this the experimentation there was no remarkable differences. I didn't try in this assigment but it is also possible to optimize the learning rate and the momentum.

II.4 dropout

I decided to remove the dropout to see the impact. The dropout allows to avoid overfitting by freezing a certain percentage of weight at each iteration (for forward and backpropagation).

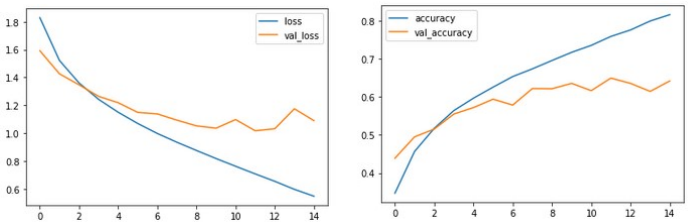


FIGURE 4 – model without dropout

We can see that the performances are very bad and that from 2 epochs onwards the model is already overfitting because val_loss is worse than loss and start to increase. So it is necessarity to use dropout the percentage of dropout is given in figure 1.

II.5 batch normalisation

I chose to add layers for batch normalisation. This step could speed up the convergence and performance of the model. To do this, it uses mini-batches to normalise the data as the training progresses.

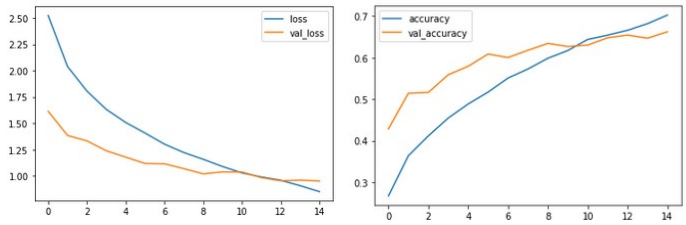
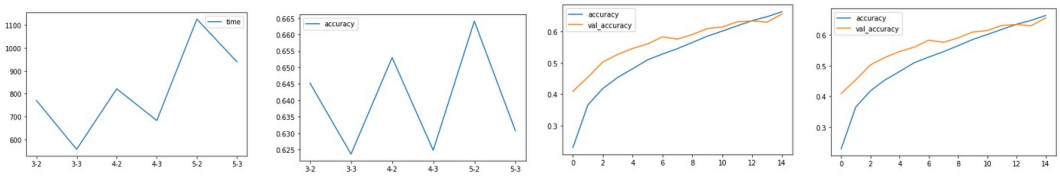


FIGURE 5 – model with batch normalisation

Here we can see that batch normalisation does not give better results and the learning curve is worse. On the other hand, I ran the scipt several times with different training set sizes, structures. Sometimes there were noticeable improvements. So, it is important to keep this in mind.

II.6 kernel size

In this part I chose to see the impact of the kernel size. The larger the kernel the more importance is given to the far surroundings for a pixel.



(a) time (b) accuracy (c) model with 5-2 (d) model with 5-3

In this test, I tested different kernel sizes for convolution and max pooling. Thus, $textbfn - p$ means convolution = (n,n) and max pooling = (p,p). For all the models, there was no overfitting and the learning curve were good. Moreover, we see that the execution time increases with the accuracy. The parameter that has the most impact is the padding kernel. We have better results when it is 2 than 3. At best we reach 66,5% for 5-2. Also, having 3-2 saves time by losing only 2%.

II.7 Add convolutional layer

Now rather than changing the base model we will look at adding layers to the base model. I will append the new structure when necessary.
In this part I have tried to add convolution layers rather than trying to change the size of the kernels or the number of neurons. (see the model structure in the Appendix IV.2.a).

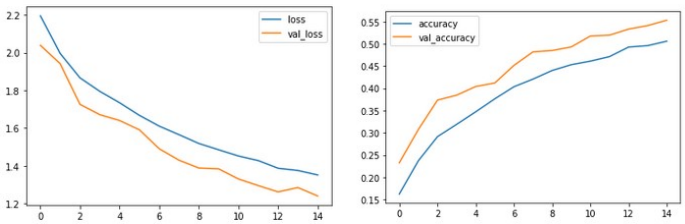


FIGURE 7 – add convolutional layer

Here, the performance is not as good because not enough data was given and the model has many weights to learn. The model would be able to learn better if it was given more data. The accuracy is around 55% which is bad compared to first model.

II.8 Add fully connceted layer

In this section, I will look at the impact of adding the fully connceted layer. I put the structure in append IV.2.b.

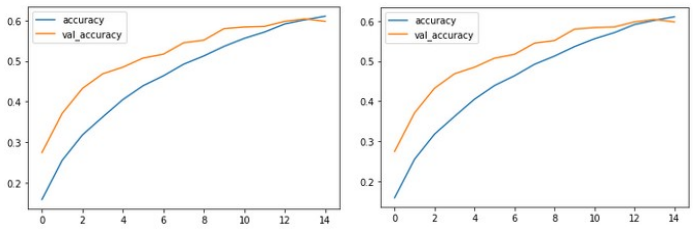


FIGURE 8 – modèle de référence

The performance is equivalent or even worse than the original models. However, the execution time is more important. So I don't think adding more fully connected will improve performance.

II.9 batch size

I will study the impact of batch size on performance and the impact on time.

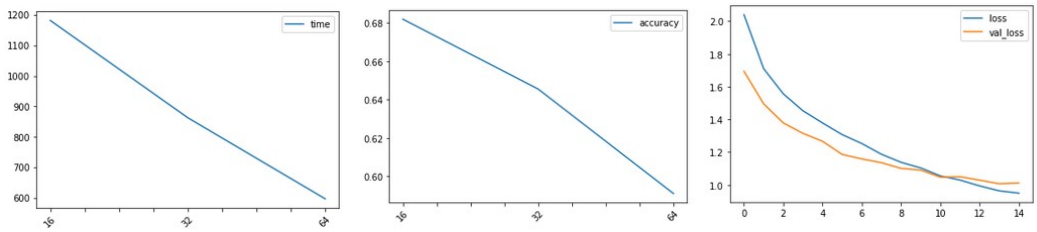


FIGURE 9 – performance versus batch size

The smaller the batch, the longer the model but the better the performance in accuracy. The figure on the right is model with batch size = 16. You can see it is not overfitting. If the batch is too small, it will allow overfitting, while if it is too small, the model have difficulties to learn.

II.10 number of neurons per layers

I will study the impact of the number of neurons per layer on performance and the impact on time. The legend explain how to understand the notation (in the caption) in x -axis. I put the structure for the first one (32 - 64 - 32 - 64) in appendix IV.2.c.

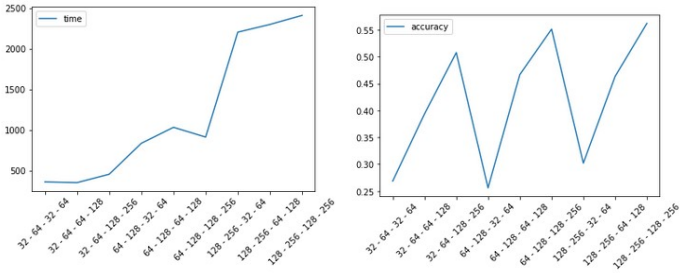


FIGURE 10 – legend : n_conv1 - n_conv2 - n_fully1 - n_fully2

Here, we see that the most important parameter is to add more neurons for the fully connected layers. Also, we don't have a big increase in performance when we vary the number of neurons in the convolutional part (at constant number of neurons in the deep forward part). On the other hand, the addition of neurons in the fully connected part is beneficial for learning. Logically, the more neurons one has, the longer the time.

II.11 Proposal of an optimised model

Using all the previous results and considerations, I will try to propose a model that is the most efficient. I am aware that I have studied the parameters one after the other. Thus, it is possible that combinations of parameters that are not optimal locally are optimal globally. I made this choice because I felt that this was the spirit of the assignment and I wanted to understand the impact independently of all the parameters. This allows me to gain understanding and intuition. This will be super useful for my next experiments. One of the most important parameters is the dropout. In our case I found that focusing on the neurons at the deep forward level rather than the convolutional part gave better improvements. I chose to use kernel = (2,2) for the padding and kernel = (3,3) for the convolution. Otherwise for the number of neurons I did according to the best accuracy in II.10. I chose a batch size of 32 and 15 epochs. This time I used the whole training set and I don't use batch normalisation. I put this final model in appendix IV.3. I will also put the result with batch normalisation in appendix IV.4 if you want to compare.

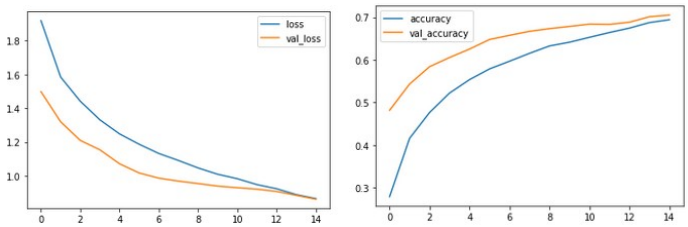


FIGURE 11

With this set-up, I reach 73% which is better than before.

III Conclusion

In conclusion, this assignment allowed to see the importance of each part. Also it helps to understand how to build a model from scratch. The method I would use now would be to find a model that worked on a similar domain such as vgg, darkNet, resnet. First I will try to do transfer learning if it was applyed to similar topics. If this is not possible I will try to train the model with initial random weight. I will also try to add dropout and/or batch normalisation. Finally, playing with the number of convolution layers and the number of neurons for the fully connected part. Typically what was working here could not work on other dataset so it is a case to case optimisation. Also, it is important to size a model according to the amount of data available. That could justify data augmentation. I was able to experiment with the benefits of this on another project. We also could see that it was complicated to find a model that reached performances around 80%. Comparing on the internet, I noticed that most people were getting from 70% to 75% at best. The only model I found that was over 85% used data augmentation as well. As here we have limited our training set so I did not consider this method. We can also consider the stride and the optimizer optimisation.

.

IV APPENDIX

IV.1 Images

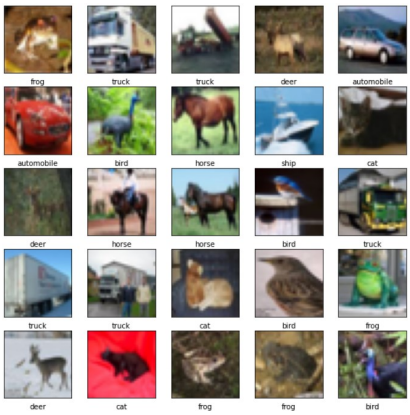


FIGURE 12 – dataset images

IV.2 Intermediate models structures

Layer (type)	Output Shape	Param #
conv2d_185 (Conv2D)	(None, 32, 32, 32)	896
conv2d_186 (Conv2D)	(None, 32, 32, 32)	9248
max_pooling2d_87 (MaxPooling2D)	(None, 16, 16, 32)	0
dropout_154 (Dropout)	(None, 16, 16, 32)	0
conv2d_187 (Conv2D)	(None, 16, 16, 64)	18496
conv2d_188 (Conv2D)	(None, 16, 16, 64)	36928
max_pooling2d_88 (MaxPooling2D)	(None, 8, 8, 64)	0
dropout_155 (Dropout)	(None, 8, 8, 64)	0
flatten_48 (Flatten)	(None, 4096)	0
dense_126 (Dense)	(None, 256)	1048832
dropout_156 (Dropout)	(None, 256)	0
dense_127 (Dense)	(None, 256)	65792
dropout_157 (Dropout)	(None, 256)	0
dense_128 (Dense)	(None, 10)	2570

Total params: 1,182,762
Trainable params: 1,182,762
Non-trainable params: 0

(a) model add fully convolutional layers

Layer (type)	Output Shape	Param #
conv2d_191 (Conv2D)	(None, 32, 32, 32)	896
conv2d_192 (Conv2D)	(None, 32, 32, 32)	9248
max_pooling2d_90 (MaxPooling2D)	(None, 16, 16, 32)	0
dropout_162 (Dropout)	(None, 16, 16, 32)	0
flatten_50 (Flatten)	(None, 8192)	0
dense_133 (Dense)	(None, 256)	2097408
dropout_163 (Dropout)	(None, 256)	0
dense_134 (Dense)	(None, 256)	65792
dropout_164 (Dropout)	(None, 256)	0
dense_135 (Dense)	(None, 256)	65792
dropout_165 (Dropout)	(None, 256)	0
dense_136 (Dense)	(None, 10)	2570

Total params: 2,241,706
Trainable params: 2,241,706
Non-trainable params: 0

(b) model add fully connected layer2

Model: "sequential_58"

Layer (type)	Output Shape	Param #
conv2d_193 (Conv2D)	(None, 32, 32, 128)	3584
conv2d_194 (Conv2D)	(None, 32, 32, 256)	295168
max_pooling2d_91 (MaxPooling2D)	(None, 16, 16, 256)	0
dropout_166 (Dropout)	(None, 16, 16, 256)	0
flatten_51 (Flatten)	(None, 65536)	0
dense_137 (Dense)	(None, 32)	2097184
dropout_167 (Dropout)	(None, 32)	0
dense_138 (Dense)	(None, 64)	2112
dropout_168 (Dropout)	(None, 64)	0
dense_139 (Dense)	(None, 10)	650

Total params: 2,398,698
Trainable params: 2,398,698
Non-trainable params: 0

(c) model add neurons

IV.3 Best model

Layer (type)	Output Shape	Param #
conv2d_175 (Conv2D)	(None, 32, 32, 64)	1792
batch_normalization_33 (BatchNormalization)	(None, 32, 32, 64)	256
conv2d_176 (Conv2D)	(None, 32, 32, 128)	73856
batch_normalization_34 (BatchNormalization)	(None, 32, 32, 128)	512
max_pooling2d_82 (MaxPooling2D)	(None, 16, 16, 128)	0
dropout_143 (Dropout)	(None, 16, 16, 128)	0
flatten_45 (Flatten)	(None, 32768)	0
dense_117 (Dense)	(None, 128)	4194432
batch_normalization_35 (BatchNormalization)	(None, 128)	512
dropout_144 (Dropout)	(None, 128)	0
dense_118 (Dense)	(None, 256)	33024
batch_normalization_36 (BatchNormalization)	(None, 256)	1024
dropout_145 (Dropout)	(None, 256)	0
dense_119 (Dense)	(None, 10)	2570

Total params: 4,307,978
Trainable params: 4,306,826
Non-trainable params: 1,152

FIGURE 14

IV.4 Best model without Batch

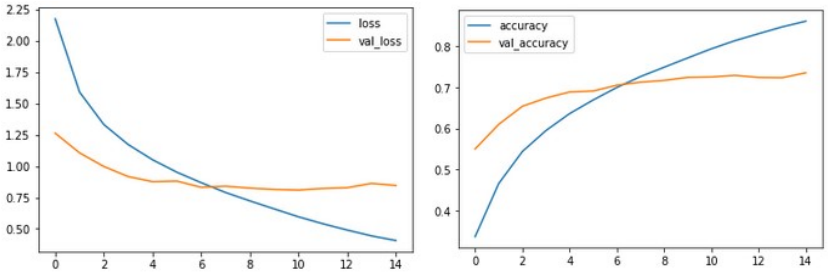


FIGURE 15 – final model