RDFIA - Convolutional Neural Networks (1-c and 1-d)

Bourzag Mohamed Chakib, Missoum Youcef

October 2025

1 Introduction

This document contains the exercises and written solutions for the CNN practical work (RDFIA TP1-c/d).

Exercise 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 $x \times y \times z$:*

   Solution.

   - ***Output Height*** H'*:*
   $$H' = \frac{x - k + 2p}{s} + 1$$
   - ***Output Width*** W'*:*
   $$W' = \frac{y - k + 2p}{s} + 1$$
   - *Since we have only one filter, one dimension (the number of channels, here we consider it z) would disappear because we sum along the third dimension. So, we only would have a 2D output image.*
   - *There is $(k * k * z + 1)$ parameters to learn, if we consider the dimension z as the number of channels (filter size * number of channels).*
   - *With a fully connected layer, we would need $(x * y * z + 1) * (H' * W')$, because we need to give the whole image as an input to a neuron (with biais) and have $H' * W'$ neurons corresponding to the output image size.*

   □

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

   Solution.
   Advantages*:*

   - ***Parameter sharing****: leads fewer weights that can be shared across different nodes.*
   - ***Easier training****: since it involves less parameters.*
   - ***Sparse connectivity****: helps to learn learning local patterns and extract image features.*
   - ***Stacking layers****: CNNs can learn complex relationships between parts.*

   Limit*:*

   - *CNNs have a limited ability to capture spatial hierarchies in a robust, transformation-invariant way. They can be sensitive to variations such as rotation, scale, or viewpoint unless extensive data augmentation or architectural modifications (e.g., spatial transformer networks) are applied.*
   - ***Limited global context understanding****: CNNs use local receptive fields and thus mainly capture local features. Even though deeper layers aggregate larger contexts, CNNs may still struggle to model long-range dependencies or global spatial relationships — a limitation that motivated architectures like Transformers and Vision Transformers (ViTs).*
   - ***Computational cost****: CNNs can be computationally expensive to train and deploy, especially on high-resolution inputs or with very deep architectures.*

   □

3. *Why do we use spatial pooling?*

   Solution.

- ***Dimensionality Reduction:*** *reduces image size (H and W) which decreases the computational load and the number of parameters in subsequent layers.*
- ***Translation Invariance:*** *invariance to small shifts and distortions of the image.*

□

4. *Suppose we try to compute the output of a classical convolutional network for an input image larger than planned (e.g., 224×224). Can we use all or part of the layers?*

   Solution.
   Yes, we can use the convolutional and pooling layers on an image of any size but for FC layers expect a fixed-size input vector. So it depends on feature map from the last conv/pool layer, the flattened vector fed to the FC layer will have the wrong dimensions causing an error if we don't manage that. □

5. *How can we modify a convolution filter to make it equivalent to a fully-connected layer for one central output pixel?*

   Solution.
   By making the kernel size of the convolutional filter equal to the spatial dimensions of the input feature map. For example, if the input feature map is 16x16x32, so we use a convolutional filter of size 16x16, with padding = 0 and stride = 1. □

6. *For an unshared convolution layer with input (B, C, H, W) and Q filters of size 5×5, how many parameters (excluding biases) are there?*

   Solution.
   An "unshared" convolution *is different from a standard one because it does not share weights across different spatial locations. It has a unique set of weights for every input pixel.*

   So, the number of parameters would be:

   $$(5 \times 5 \times C) \times H \times W$$

   □

7. *Define the receptive field of a neuron. What are the receptive field sizes for the first and second convolutional layers? How do they evolve in deeper layers?*

   Solution.

   - *A receptive field is the specific part or region of the original image that a neuron can see and get influenced by it.*
   - *The receptive field size of first layer is the* ***kernel size*** *k_1. So, $RF_1 = k_1$.*
   - *For the second layer, it would be: $RF_2 = RF_1 + (k_2 - 1)$.*
   - *A more general formula would be:*

     $$RF_l = RF_{l-1} + (k_l - 1) * jump_l$$

     with:

     $$jump_l = jump_{l-1} * stride_l,\ jump_0 = 1$$

   - *As we go deeper into the layers, the receptive field of a neuron becomes larger than in the previous layers because each neuron aggregates information from a larger part of the input via multiple layers. As a result, the network would be able to have a more global perspective of the image as we go deeper from layer to layer.*

   □

Exercise 2. ***Training the Model from Scratch***

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

  Solution.

  - *padding = $\frac{k-1}{2}$, k is the size of kernel*
  - *So padding =2, stride=1 for a 5*5 conv*

  □

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

  Solution.

  - $$Output_W = \frac{W - k + 2p}{s} + 1 \quad \longrightarrow \quad \frac{W - 2 + 2(0)}{2} + 1 \quad \longrightarrow \quad \frac{W}{2} - 1 + 1 \quad \longrightarrow \quad \frac{W}{2}$$
  - *So padding =0, stride=2 for a 2*2 maxpool*

  □

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

  Solution.

  - *output size = vector of 1*10*
  - *conv1 = (5*5*3 + 1) * 32 = 2,432*
  - *conv2 = (5*5*32 + 1) * 64 = 51,264*
  - *conv3 = (5*5*64 +1) * 64 = 102,464*
  - *fc4 = (1024+1)*1000 = 1,025,000*
  - *fc5 = (1000+1)*10 = 10,010*

  We can see that the number of parameters in the fc4 is extremely big compared to the others. This is due to the large number of output filters of the last convolution layer. □

- *11. What is the total number of weights to learn? Compare it to the number of examples.*

  Solution.

  - *Total numbers of weights = 1,191,170*
  - *Comparison: total numbers of weights >>>> number of examples (weights are Much larger than examples). This may cause severe **overfitting**.*

  □

- *12. Compare the number of parameters to learn with those of the BoW and SVM approaches.*

  Solution.

  SVM parameters:*The number of parameters would depend on the size of the "visual vocabulary" and the number of support vectors in the SVM. For example, a BoW model with a vocabulary of 500 visual words and a linear SVM classifier for 10 classes would have roughly (500 + 1) * 10 = 5,010 parameters.*

  Conclusion: *The CNN is much more more parameter-heavy and expressive.* □

- *14. In the provided code, what is the major difference between how loss and accuracy are computed during training and testing?*

  Solution. *The major difference lies in updating or not the network parameters. When the optimizer is given (along with training data), the code performs a training epoch (model.train()) updating the parameters using the calculated loss. In contrast, when the optimizer is no given as an input, the code performs an evaluation epoch using the test data, calculating the loss and the accuracy without updating the model parameters (model.eval()).* □

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

  Solution.

  - *learning rate: defines how much of the loss value is used to update the parameters. Smaller learning rate would most likely lead to a good convergence but slower training and might stuck the parameters in a local optima (if the loss function is not convex). In contrast, a bigger learning rate would lead to a faster training, might avoid the local optima problem but might cause instability and oscillations during training.*
  - *batch size: the number of samples along which to update the parameters. It defines how many times or how fast the parameters are updated. The loss is accumulated along the "batch size" samples and the parameters are only updated when the whole batch is traversed. If the batch size is big, the parameters would be updated after a long time, leading to a slower convergence but a faster training. In contrast, if the batch size is small, the parameters would be updated more frequently, leading to a better convergence but a slower training.*

□

- *17. What is the error at the start of the first epoch (train/test)? How can you interpret it?*

  Solution.

  - *the error at the start of the first epoch (train/test) is:*
    - *train: 2.3020*
    - *test: 1.9914*
  - *The model is just at the beginning of training, but having a smaller test error is probably due to the random initialization of parameters that could capture more insights of the test data than training data.*

□

- *18. Interpret the results. What's wrong? What is this phenomenon?*

  Solution. *We trained the model for 50 epochs. The training loss converges to 0 while the test loss after a certain number of epochs starts increasing. This is known as the overfitting phenomenon, where the model learns too well on the training and looses generalization capacity.* □

Exercise 3. ***Improving the Results***

3.1. Standardization of examples

- *19. Describe your experimental results with standardization.*

  Solution. *We notice a really small improvement but there is still the overfitting problem. In addition, at a certain level, the training just diverges and accuracies goes down to 0, showing a high instability.* □

- *20. Why compute the average image only on the training set and use it for validation normalization?*

  Solution.

  - *We use only mean and std from the training set because the validation and test sets must simulate new, unseen data. We cannot use any statistics derived from them during training — otherwise, we leak information from the validation/test set into the training process (data leakage) and bias the output. Moreover, since test data is unseen data, recalculating means and stds during inference would take longer time.*

□

- *21.* ***Bonus :*** *We tried to explore ZCA normalization and minmax normalization. The first could only be executed for few epochs (due to GPU limitations) but showed at the beginning a quite stable learning compared to standization (but still too early to make a judge).*

3.2. Increase in the number of training examples by data increase

- *22. Describe your experimental results with data augmentation and compare them to previous results.*

  Solution. *Trained it for 50 epochs (with and without normalization) and we noticed a good improvement compared to previous cases since test loss nearly stabilizes and doesn't increase that much. Nevertheless, we still have some overfitting.* □

- *23. Discuss when horizontal symmetry (flip) is or is not usable.*

  Solution.

  - *It can be useful when dealing with normal objects as it gives a different perspective to it.*
  - *However, it may cause confusion or change the output's meaning with certain entities like numbers or letters, where changing the rotation completely changes the meaning.*

□

- *24. What are the limits of data augmentation by transformations?*

  Solution.

  - *It has a strong relation with the original picture. It does not create new information other than new slightly different forms.*
  - *It does not always keep semantics. It may lead to confusions in certain cases (letters, numbers...) where the shape matters.*
  - *Not always representative of real-world test data.*

□

- *25.* ***Bonus :*** *We noticed that adding more transformations such as rotation and color jitter (which randomly changes brightness, contrast, saturation, and hue) stabilized training more.*

3.3. Variants on the optimization algorithm

- *26. Describe your experimental results when using a learning rate scheduler.*

  Solution.

  - *When using it alone without normalization and data augmentation, we noticed a really fast convergence at nearly 20 epochs on the training set but there a big overfitting as test loss increases simultaneously after nearly 10 epochs.*
  - *When keeping data augmentation transformations this time (same as before: RandomCrop and RandomHorizontalFlip), we noticed a more stable training than when using transformations alone, where the training error keeps decreasing and the test error stabilizes at a certain point without increasing.*
  - *When adding normalization to this last one (applied before transforming), we noticed a slightly bigger overfitting as test loss increases slowly.*

□

- *27. Why does this method improve learning?*

  Solution.

  - *A learning rate scheduler improves learning by adapting the step size during training. At the beginning, a larger learning rate helps the optimizer explore the loss surface and escape poor local minima. As training progresses, reducing the learning rate allows for finer adjustments near the optimal solution, preventing oscillations and overshooting. This dynamic adjustment leads to faster convergence, greater stability, and often better generalization than using a fixed learning rate.*

□

- *28.* ***Bonus:***

  - *Many variants of SGD exist, including* ***Momentum***, ***Nesterov Accelerated Gradient***, ***Adagrad***, ***RMSProp***, ***Adam***, *and* ***AdamW***. *These methods aim to adapt the step size or direction of updates to accelerate convergence and stabilize learning. The table 1 summarizes some of those techniques:*

Optimizer	Key Idea	Notes
SGD (vanilla)	Updates weights using gradient of loss w.r.t. parameters.	Simple but can get stuck in local minima or oscillate.
SGD with Momentum	Adds a "velocity" term that accumulates past gradients.	Helps escape shallow minima and accelerates convergence.
Nesterov Accelerated Gradient (NAG)	Similar to momentum, but looks ahead before computing gradient.	More responsive and slightly faster than standard momentum.
Adagrad	Adapts the learning rate per parameter based on gradient history (large updates for rare features).	Often used in NLP; learning rate shrinks too much over time.
RMSProp	Keeps a moving average of squared gradients to normalize learning rate.	Works well for non-stationary problems (RNNs).
Adam (Adaptive Moment Estimation)	Combines Momentum + RMSProp: uses both first and second moment estimates.	Very popular; usually faster convergence.
AdamW	Adam with decoupled weight decay (better regularization).	Common in CNNs and Transformers.
AdaBelief / AdaBound	Variants that adapt step sizes more carefully.	More stable than Adam in some cases.

Table 1: Comparison of Common Optimization Algorithms

  - *Similarly, several learning rate scheduling strategies exist such as* ***Exponential Decay*** *(the one we previously used),* ***Step Decay***, ***Cosine Annealing***, ***Cyclic Learning Rate***, *and* ***One-Cycle Policy***. *These methods dynamically adjust the learning rate during training to improve convergence and avoid local minima. The table 2 below summarizes some of those techniques:*
  - *In practice, we previously tested* ***SGD*** *with* ***ExpoLR***, *in addition to* ***Adam*** *with* ***CosineAnnealingLR***, *and* ***AdamW*** *with* ***OneCycleLR***, *observing that adaptive optimizers with smooth LR schedules led to faster and more stable convergence on CIFAR-10 if the right combination is choosen, for instance:*
    - ***Adam*** *with* ***CosineAnnealingLR*** *led to severe overfitting.*
    - ***AdamW*** *with* ***OneCycleLR*** *showed some instability and overfitting.*

Strategy	Description	Example
Constant LR	Same value throughout training.	`lr = 0.01`
Step Decay	LR decreases by a factor every few epochs.	`lr *= 0.1 every 30 epochs`
Exponential Decay	LR decays continuously with an exponential function.	`lr = lr0 * exp(-k * epoch)`
Cosine Annealing	LR follows a cosine curve between max and min values.	Smooth decay, used in modern CNNs.
One-Cycle Policy	LR first increases, then decreases (used in *fastai*, PyTorch).	Accelerates convergence, prevents bad minima.
Cyclic LR	LR oscillates between lower and upper bounds.	Helps escape local minima.
Warmup	Gradually increases LR at the start before decaying.	Common with large models (ResNet, Transformers).

Table 2: Learning Rate Scheduling Strategies

3.4. Regularization of the network by dropout

- *29. Describe your experimental results after adding dropout.*

  Solution. *We tested on all configurations using (or not) the techniques covered until now, including normalization, transformation data augmentation, learning rate scheduler and finally the dropout (p=0.5). We trained our model for 50 epochs on each possible configuration and the best configuration included the dropout, transformations and learning rate scheduler without normalization. This configuration presented the most stable training without overfitting and it minimized the most both training and test losses.* □

- *30. What is regularization?*

  Solution. *Regularization refers to any technique that aims to reduce overfitting by penalizing model complexity or introducing constraints during training. In other words, it helps the model generalize better to unseen data instead of memorizing the training set.* □

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

  Solution. *Dropout can be interpreted as a regularization technique that improves the generalization ability of neural networks. During training, each neuron is randomly "dropped" (set to zero) with a certain probability, which prevents the network from relying too heavily on specific neurons or feature combinations. This stochastic behavior forces the network to learn more robust, distributed representations of the data.*

  From another point of view, dropout can be seen as implicitly training an ensemble of many smaller sub-networks that share weights. At test time, the full network behaves like the average prediction of this ensemble, which reduces variance and limits overfitting.

  Additionally, dropout acts as a form of noise injection into the intermediate representations, encouraging the model to become invariant to small perturbations and making it more stable and resilient to unseen data. □

- *32. What is the influence of the hyperparameter of this layer ?*

  Solution. *The drop probability* $\boldsymbol{p}$ *controls the regularization strength: small values (e.g., 0.1–0.3) give mild regularization, while large values (e.g., 0.5) enforce stronger regularization but can lead to underfitting if too high. Thus,* $\boldsymbol{p}$ *directly influences the balance between* ***bias*** *and* ***variance****: low* $\boldsymbol{p}$ *leads to lower bias but higher variance (overfitting risk), while high* $\boldsymbol{p}$ *increases bias but reduces variance (underfitting risk).* □

- *33. What is the difference in dropout behavior between training and test phases?*

  Solution.

  - *Each neuron is randomly deactivated with probability p. The surviving neurons' activations are scaled by a factor* $1/(1-p)$ *so that the expected total activation remains the same.*
  - ***During test (or inference):*** *Dropout is disabled — all neurons are active. No random dropping occurs, and activations are used as-is (since they were already scaled during training). This ensures that the network's output is deterministic and that its expected magnitude matches the training phase.*

  □

3.5. Use of batch normalization

- *34. Describe your experimental results after applying batch normalization.*

  Solution. *Having multiple possible configurations as in the previous questions, we kept the best configuration from the previous part (3.4) and trained it for 50 epochs.It showed some instability during training but it would*

converge as the epochs go, showing more and more stability at the end without overfitting. Training it for more would be more interesting. □

Conclusion:

To conclude, this work shows that successful training requires extensive hyperparameter tuning and careful architecture exploration to identify the most effective model. We observed that using SGD with an exponential learning rate scheduler provides solid performance, especially when combined with data augmentation, dropout, and batch normalization, which together promote stable and robust learning.

However, further experiments are still needed — particularly those combining different normalization schemes and learning rate scheduling techniques introduced in the bonus section. Such exploration could yield additional insights and potentially lead to an even better-performing model, illustrating that model optimization is an iterative and often lengthy process.