

INF265

Project 2:

Object Detection

Deadline: March 25th, 23.59

Deliver here:

<https://mitt.uib.no/courses/33548/assignments/57657>

Projects are a compulsory part of the course. This project contributes a total of 15% of the final grade. **The project is done in pairs. However, you are allowed to do it alone if you cannot find a pair.** If you do the work with a pair, add a paragraph to your report explaining the division of labor (Note that both students will get the same grade regardless of the division of labor).

Code of conduct: You are allowed to copy-paste code from the solutions of previous weekly exercises. However, it is not allowed to copy-paste from online sources nor from other students' projects. To some extent, discussions on parts of the project with other pairs/students are tolerated. If you do so, indicate with whom and on which parts of the project you have collaborated. Section 2.2 provides some hints, if you need additional assistance, teaching assistants and group leaders are available to help you.

Grading: Grading will be based on the following qualities:

- Correctness (your answers/code are correct and clear)
- Clarity of code (documentation, naming of variables, logical formatting)
- Reporting (thoroughness and clarity of the report)

Deliverables: You should deliver 2 files:

- A jupyter notebook addressing the tasks defined in this project. Cells should already be run and output visible.
- A PDF report addressing section 4. Note that exporting your notebook as a PDF is not what is expected here. Each point described in section 4 has to be done for both the object localization and the object detection tasks. Remember to include the plots that are expected.

If you need to provide additional files, include a `README.txt` that briefly explains the purpose of these additional files. In any case, do not include the datasets in your submission.

Late submission policy: All late submissions will get a deduction of 2 points. In addition, there is a 2-point deduction for every starting 12-hour period. That is, a project submitted at 00.01 on March 26th will get a 4-point deduction and a project submitted at 12.01 on the same day will get a 6-point deduction (and so on). All projects submitted on March 28th or later are automatically failed. (Executive summary: Submit your project on time.) There will be no possibility to resubmit failed projects so start working early.

1 Introduction

In this project, you will define and train convolutional neural networks to solve an object localization task and an object detection task.

To clearly distinguish between the challenges related to the classification of an object and the definition of its bounding box and the challenges of having more than one object per image, section 2 assumes that there is at most one object per image while section 3 allows for more than one object per image.

Objectives include getting a better understanding of **a)** convolutional neural networks **b)** object localization and **c)** object detection tasks.

In this project, an augmented version of MNIST is provided and is such that:

- image dimension $H_{in} \times W_{in}$ are 48×60 .
- digits are randomly located in the image
- digits are randomly slightly rotated
- digits are randomly slightly resized (smaller or larger)
- random noise is added in the background

To each section of this project corresponds a set of train/validation/test dataset, more information about each set is provided in their corresponding section.

2 Object localization

Object localization consists in classifying an image and drawing a bounding box around the instance present in the image. It assumes that there is at most one class instance per image. In order to learn how to draw this bounding box (bb), the output layer of the neural network, the loss function and the performance measures have to be adjusted.

Output layer

In image classification, the sole task of a neural network is to associate an image with exactly one label. The output layer has then C components, one for each class $[c_1, \dots, c_C]$.

In object localization, the neural network still has to associate an image with a class, but, in addition, it has to define a bounding box (bb) around the object. The output layer has now $C + 5$ components: $[p_c, x, y, w, h, c_1, \dots, c_C]$. The first element p_c represents a binary class indicating whether there is an object or not in the image. The 4 components x, y, h, w define the bounding box (bb). (x, y) represents the coordinates of the center of the bb, with both x and y between 0 and 1. The point $(0, 0)$ is the top left corner of the image and $(1, 1)$ the bottom right corner. w and h respectively stand for *width* and *height* of the bb, they are also between 0 and 1.

In practice, there are differences between the expected output `y_true` and the predicted output `y_pred`. First, p_c is binary in `y_true` but continuous in `y_pred`. Secondly, there are C (continuous) components in `y_pred` to represent each class, while in `y_true`, the true label c is encoded as an integer between 0 and $C - 1$. Therefore, `y_true` has only 6 components: $[p_c, x, y, w, h, c]$ while `y_pred` has $C + 5$ components.

Loss function

In image classification, the loss function is usually the cross entropy loss, (`nn.CrossEntropyLoss` in pytorch), or equivalently the negative log likelihood loss of the logsoftmax of the output layer, (a combination of `F.log_softmax` and `nn.NNLoss`).

In object localization, the loss function is split into 3 components:

- A detection loss L_A : "*Is there an object in the image?*". It is defined as the binary cross entropy loss of the sigmoid of p_c . (`nn.BCEWithLogitsLoss` in pytorch, or a combination of `F.sigmoid` and `nn.BCELoss`)
- A localization loss L_B : "*Where is the object?*". This can simply be the mean squared error loss applied to $[x, y, w, h]$ (`nn.MSELoss` in pytorch)
- A classification loss L_C : "*Which class is this?*". This is the same as the image classification loss, applied to $[c_1, \dots, c_C]$.

If the expected output `y_true` is such that $p_c = 0$ (i.e. there is no object in the image), then the loss function $L_{\text{localization}}$ is reduced to the detection loss: $L_{\text{localization}} = L_A$.

If the expected output `y_true` is such that $p_c = 1$ (i.e. there is an object in the image), then the loss function $L_{\text{localization}}$ is the sum of the 3 losses: $L_{\text{localization}} = L_A + L_B + L_C$.

Performance

In image classification, the most commonly used performance is the accuracy. In object localization, accuracy is still relevant, but not sufficient for it does not tell how well the model predicts bounding boxes. In addition, its definition is slightly different as there might be images without any object at all. The total number of images to classify is the total number of images containing an object (i.e. when the first element p_c of `y_true` is 1). The number of correctly classified images, is the number of predictions where both the predicted p_c and the predicted label matches their counterparts in `y_true`. Note that interpreting the predicted p_c depends on the choice of implementation of the detection loss L_A . If the sigmoid is included in the neural network as an activation function, then an object is considered detected if $p_c > 0.5$. Otherwise, if the sigmoid is included in the loss function, then an object is considered detected if $\text{sigmoid}(p_c) > 0.5$.

Performance on bounding boxes can be measured using intersection over union (IoU). This is defined as the ratio between the area of the intersection of the box predicted and the box expected and the area of their union. IoU is then also defined between 0 and 1.

To evaluate how well the model can classify and draw bounding boxes, an overall performance can be defined as the mean of the accuracy and the IoU.

2.1 Tasks

- Load the 3 localization datasets `localization_XXX.pt`. There is at most one digit per image. All digits are represented ($C = 10$).
- Implement and train several convolutional models suitable for an object localization task and the data provided.
- Select the best model based on its overall performance.
- Evaluate the best model.
- Plot some of the images of the datasets and draw the predicted and true bounding boxes. Print their true and predicted labels as well.

2.2 Hints

- You can implement/get inspiration from the LeNet5 model architecture.
- In pytorch, the loss function is just a regular function. You can define a custom loss function by defining a regular python function and use a combination of pre-defined pytorch loss functions inside this custom function.
- In pytorch, all elements of a given tensor share the same type. But we saw that the last element of `y_true` is supposed to be an integer. You might need to convert some elements of your tensors to a different type *on the fly* while computing the loss.

- To make conditional computations more efficient, you might want to use `torch.where`.
- To draw bounding boxes, simply change the pixel values of the image at the expected location of the bounding boxes. You can use green pixel values for true bounding boxes and red pixel values for predicted bounding boxes.

3 Object Detection

Object detection is a generalization of an object localization task that allows for more than one object per image. The task then consists in classifying each object present in the image and drawing a bounding box around each of them. The image can be divided into multiple cells, of small enough size so that it becomes reasonable to assume that there is only one object per cell. An object is considered inside a cell if its center (x, y) is inside the cell (its bounding box can go beyond the limit of the cell). Once the image has been divided into $H_{out} \times W_{out}$ cells, the object detection problem is reduced to $H_{out} \times W_{out}$ independent object localization problems.

A right balance has to be found on the number of cells: on one hand, the smaller the grid cells, the more reasonable it is to assume that there is only one object per cell, but on the other hand, the smaller the grid cells, the higher the computational cost.

The division of the image into smaller grid cells requires the entire neural network architecture, the expected output as well as the loss function to be adjusted.

Architecture

In object localization with C different classes, the output layer is composed of $C + 5$ elements $[p_c, x, y, w, h, c_1, \dots, c_C]$ in order to predict both the class and the bounding box.

In object detection, the neural network has to solve a localization task for each of the $H_{out} \times W_{out}$ grid cells. The output layer has then $H_{out} \times W_{out} \times (C + 5)$ components. It can be seen as the $H_{out} \times W_{out}$ output of a convolutional layer, with $(C + 5)$ channels. But the output dimension of a convolutional layer depends on the dimension of its input, which itself depends on the previous convolutional layer. Therefore, the entire architecture of the convolutional neural network must be designed such that the input image of dimension $H_{in} \times W_{in}$ yields an output of dimension $H_{out} \times W_{out}$. This also implies that all fully connected layers have to be converted into convolutional layers.

Expected output

The expected output is initially independent of the number of grid cells decided. For each image, it is a list of $[p_c, x, y, w, h, c]$ tensors, the length of the list corresponding to the number of digits in the image. Once a $H_{out} \times W_{out}$ grid is drawn, the lists of $[p_c, x, y, w, h, c]$ tensors have to be converted into tensors of dimension $H_{out} \times W_{out} \times 6$. This adds a preprocessing step in the object detection pipeline, compared to an object localization task.

For a given image, let's denote `list_y_true` the list of $[p_c, x, y, w, h, c]$ tensors and `y_true` the processed expected output in the $H_{out} \times W_{out}$ grid. In `list_y_true`, x, y, w, h are all within $[0, 1]$ range with $(0, 0)$ still being the top left corner of the image and $(1, 1)$ the bottom right corner. But in `y_true`, x, y, w, h are defined locally, using as frame of reference the grid cell there are in. $(0, 0)$ and $(1, 1)$ are then the corners of the *grid cell* and not of the image. In addition, h and w can now be greater than 1 if the height and/or width of the bounding box is greater than the dimensions of the grid cell.

The prediction `y_pred` also defines bounding boxes in the frame of reference of the grid cell and not in the frame of reference of the image anymore.

Loss function

The object detection loss function $L_{detection}$ is the sum of the localization loss of each grid cell:

$$\sum_{h=0}^{H_{out}-1} \sum_{w=0}^{W_{out}-1} L_{localization}[h, w]$$

Performance

In object detection, the performance is similar to the localization performance applied to each grid cell.

3.1 Tasks

- Load the 3 detection datasets `detection_XXX.pt`. There can be an arbitrary number of digits per image, the only condition is that there is no overlap between digits. To facilitate the training, only 0 and 1 are kept ($C = 2$). `y_true` is already processed for a 2×3 grid, each cell of dimension 24×20 . For the following questions, we then have $H_{out} = 2$ and $W_{out} = 3$ and `y_true` is a tensor of shape $(N_{tot}, H_{out}, W_{out}, 6)$ with N_{tot} , the total number of images in the dataset.
- Implement a model such that the fully connected layers are replaced by convolutional layers and such that the last layer yields an output of the right dimension $((N, H_{out}, W_{out}, C + 5)$ with N the batch size). You can try different variants.
- Train several models. Note that the training is harder for this task, your model might perform poorly if you can not afford a long enough training.
- Select the best model based on its overall performance.
- Evaluate the best model.
- Plot some of the images of the datasets and draw the predicted and true bounding boxes. Print their true and predicted labels as well.

3.2 Optional tasks

- Load the unprocessed expected outputs `list_y_true_XXX.pt`. They correspond to the same expected outputs as in `detection_XXX.pt` files, except that they are not processed yet. They are lists of lists of tensors of shape (6): $[p_c, x, y, w, h, c]$.
- Define a grid by choosing a value for H_{out} and W_{out} . You are allowed to choose the same values as previously $H_{out} = 2$ and $W_{out} = 3$.
- Convert `list_y_true` into a $(N_{tot}, H_{out}, W_{out}, 6)$ tensor. Remember to convert bounding boxes coordinates to local coordinates.
- If you chose different values for H_{out} and W_{out} : implement a model such that the last layer yields an output of the right dimension. Train several models. Select the best model based on its overall performance. Evaluate the best model. Plot some of the images of the datasets and draw the predicted and true bounding boxes. Print their true and predicted labels as well.

4 Report

The report should address the following points, for both the object localization problem and the object detection problem:

1. Give an explanation of your approach and design choices to help us understand how your particular implementation works.

2. Report the different models and hyper-parameters you have used.
3. Report the performance of your selected model (accuracy, IoU and the mean between of accuracy and IoU).
4. Include plots showing the true bounding boxes and label compared to the predicted bounding boxes and labels on both the training and validation dataset.
5. Comment your results. In case you do not get expected results, try to give potential reasons that would explain why your code does not work and/or your results differ.