Medical Image Analysis, handin 3

Stefano Pellegrini (MLQ211)

1 Introduction to ANN and CNN

An artificial neural network (ANN) is a set of connected artificial neurons, which can be described as a weighted directed graph, and it can be used in both regression and classification problems. The information from the *input layer* of an ANN is propagated through the network, and, in each *hidden layer*, a weighted sum of the output from the previous layer is performed. Eventually, a non linear activation function can be applied, and, ultimately, a final output is produced by the *output layer*. In order to train the parameters of the network, a cost function must be chosen. During the training process, after initialization, the weights and biases are learned through the backpropagation algorithm. Backpropagation uses gradient descent to adjust the model parameters in such a way that the cost function on the training data is minimized. There are different variations of the gradient descent algorithm, but to reduce computational complexity, the mini-batch gradient descent is often used. In this variation the training dataset is splitted into small batches, which are used to calculate the model error and update the model weights and biases according to a selected learning rate.

A convolutional neural network (CNN) is a special ANN designed to capture local spatial patterns in the input. It is featured by features extraction and spatial invariance and it is well suited for image, language, and sound processing. Its great success is partially due to its ability of exploiting massively parallel computing (e.g. GPUs) and the availability of large training datasets, which enable reducing overfitting. Generally speaking, a CNN architecture is usually composed by an *input layer*, followed by a series of *convolutional* and *pooling layers*, which are then connected to the typical layers of a standard ANN, such as one or more *hidden layers* and, lastly, the *output layer*. Typically, a rectified linear unit (ReLU) activation function is applied to the output of the convolutional operation, while, depending on the problem, different activation functions can be applied to the final output. The ReLU activation function is used because it simply outputs zero for all negative inputs, therefore it helps the training process. The *convolutional layer* performs the convolutional operation by applying one or more filters to the input, therefore performing features extraction. By applying different filters, whose parameters are learned during the training process, different features maps are generated. Each feature map shows where a certain feature is located in the input, thus allowing the networks to get rid of unnecessary information. The *pooling layer* performs the pooling operation (e.g. max, sum, or average pooling), which reduces the size of the input, preserves the extracted features, and introduces spatial invariance, which provides flexibility to the network. In order to prevent overfitting different strategies can be applied, such as regularization, dropout, and batch normalization methods.

In regularization, an extra component is added to the loss function. L1 regularization add a penalty to reduce the number of non zero parameters, while L2 regularization penalizes large parameters, inducing the network to redistribute its weights. Dropout method can reduce overfitting by randomly removing units from the neural network during the training process, in this way it is capable to approximately combines different neural network architectures. Lastly, batch normalization technique is used to stabilize the training process by stabilizing the distribution of the inputs across the network.

2 CNN implementation for image segmentation

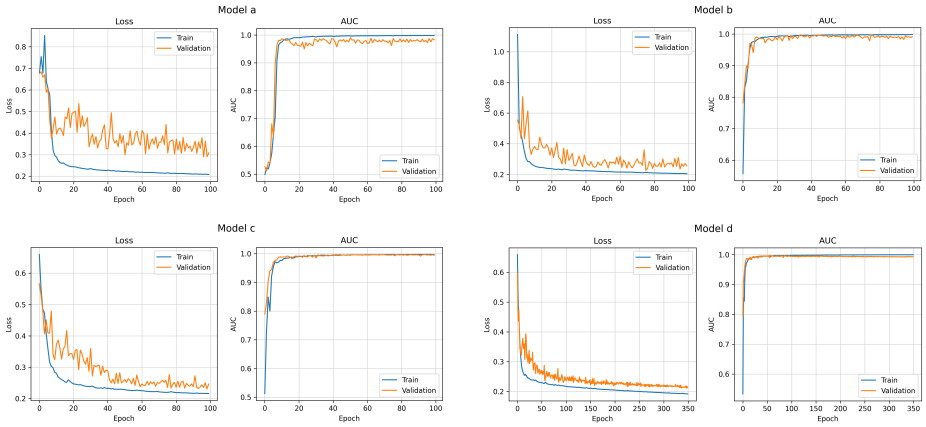

Fig. 1. Models training process. The figure shows how models performance (on training and validation data) change during the training process. Models a (top left), b (top right), c (bottom left), d (bottom right) correspond to *model a* (baseline model), *models b* and *c* (intermediate models), and *model d* (final model), whose hyperparameters are shown in Tab 1. The models are trained using the binary cross-entropy loss function (left plot of each model subfigure), but the AUC metric is also shown.
***** The different axis limits across the plots is taken into account during hyperparameters tuning, but setting the axis limit in advance could have facilitate the evaluation of the training process.

By predicting each pixel class, a CNN can be trained to perform image segmentation. Here, we performed the segmentation of lungs fields using the U-Net architecture. This CNN model takes raw x-rays lung images as inputs and, by performing a series of convolution, pooling, and up-convolution operations, it outputs the corresponding segmented lung field masks. The model parameters are learned through the optimization of a binary-cross entropy loss function. After preprocessing, we used Keras's *validation split* to perform hyperparameters tuning. By using *validation split* during the training process, at each epoch a fraction of the training data is randomly selected and is held back for validation. This allows us to monitor the training process and change the hyperparameters until the desired result is obtained. The hyperparameters we tuned to obtain

the optimal segmentation were: *batch size*, *epochs*, *learning rate*, and *validation split*. The *batch size* is the number of images that are fed into the model at each step of the training process (mini-batch gradient descent backpropagation). A small *batch size* has the advantage of speeding up the training process and using less memory resources. While a large *batch size* has the advantage of obtaining a more accurate estimate of the gradient, since more examples are used. The *epochs* represents the number of times that the entire training dataset is used to train the model parameters. A small number of *epochs* might lead to underfitting, while a large number might lead to overfitting. The *learning rate* determines the step size of the learning parameters. By choosing a *learning rate* that is too small, the model weights will take a long time to converge and they might get stuck on a suboptimal solution. On the other extreme, a too large *learning rate* will lead to parameters updates that will be too large, therefore resulting in an oscillation of the model performance over training epochs. Several different combinations of model hyperparameters were tested, but for clarity we chose only four of them. Table 1 shows the hyperparameters of the four selected models. They included the baseline model (*model a*), two intermediary models (*models b* and *c*), and the final model (*model d*). In order to obtain the hyperparameters of the final model, we observed the progress of the models performances during the training process (Fig 1), and we tuned the hyperparameters until a reasonable result was obtained. Fig 1 shows how increasing the *batch size* stabilized the learning process. Also, decreasing the *learning rate* to baseline level reduced performance oscillations across epochs. Increasing *validation split* allowed to obtain a better estimation of the validation performance. Lastly, optimal *epochs* were chosen such that the model learning ability was increased and, at the same time, overfitting was avoided (in case of overfitting, the loss on the training data would continue to decrease while the one on the validation would not).

Model	Batch size	Epochs	Learn. rate	Val. split
Model a	10	100	0.0001	0.15
Model b	20	100	0.0002	0.15
Model c	20	100	0.0001	0.15
Model d	20	350	0.0001	0.2

Table 1. Models hyperparameters. The table shows the hyperparameters of the baseline model (*model a*) and the final model (*model d*), as well as two intermediary models (*model b* and *c*). The training process of these models is shown in Fig 1.

2.1 Model performance

We assessed the model performance on the test dataset using different metrics, such as dice similarity coefficient (DSC), pixel accuracy, sensitivity (true positive rate), and specificity (true negative rate). We computed the average of these metric functions across all testing images, and the result is shown in Table 2. In order to compute the DSC and the diagnostic accuracy measures, the thresholding method was applied to both, the output of the CNN models and the ground truth masks, therefore setting all values above 0.5 as 1 and all other values as 0.

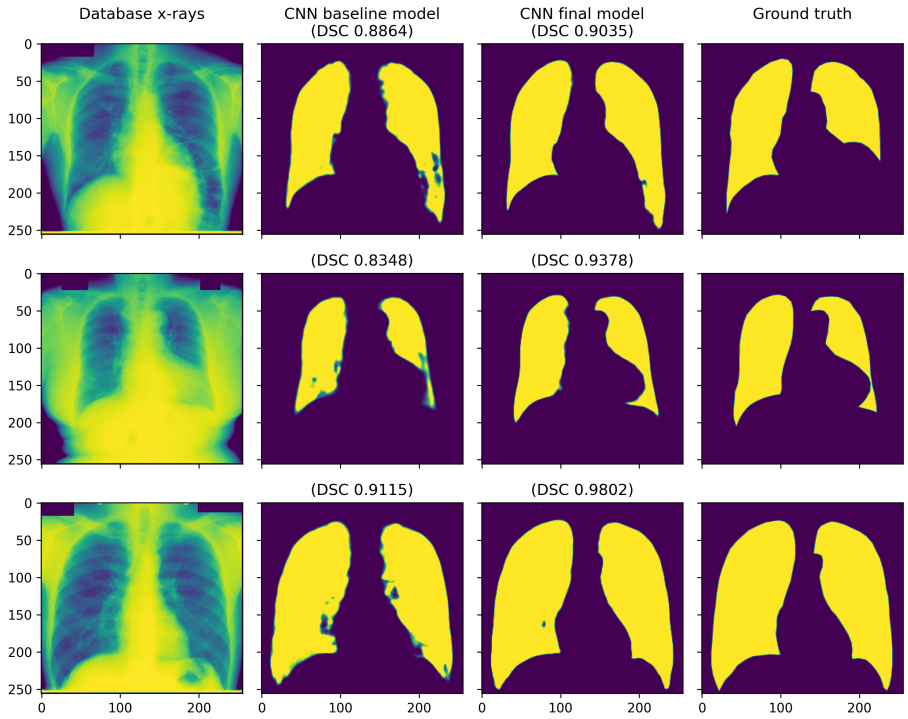

Fig. 2. Segmentation results on three testing images. The three rows respectively correspond to the segmentation of JPCLN016, JPCLN048, and JPCLN058 images. From left to right, the first column corresponds to the raw x-rays image, the second column corresponds to the segmentation obtained by the baseline model (*model a* of Tab 1), the third column corresponds to the results obtained by the final model (*model d* of Tab 1), and the fourth column corresponds to the ground truth masks. The dice similarity coefficients (DSC) obtained by the baseline and the final models are also shown (different metrics are shown in Tab 3).

The DSC is a metric specific designed for the evaluation of segmentation performance, while the others are general statistical measures of the performance of a binary classifier. We can see (Table 2) that there is a clear improvement in the generalization ability of the final model in respect to the baseline one.

Fig 2 shows the segmentation of three testing x-rays images performed by the baseline and the final models. It is possible to observe that the final model performed better than the baseline, obtaining a quite accurate segmentation of the second and third x-rays images, while both models were not able to obtain a correct segmentation of the first image. Therefore, we supposed that the segmentation of the first image was more challenging than the other two. One possible explanation is that the stomach of the patient contained some air which was predicted by the model as an extension of the right lung field. Table 3 shows

Model	DSC	Accuracy	Sensitivity	Specificity
Model a	0.8989	0.9459	0.8198	0.9994
Model d	0.9723	0.9838	0.9621	0.9929

Table 2. Models average test set performance. The table shows the average segmentation performance of the baseline model (*model a*) and the final model (*model d*) across all testing images.

Image	DSC	Accuracy	Sensitivity	Specificity
JPCLN016	0.9035	0.9477	0.9699	0.9402
JPCLN048	0.9378	0.9763	0.9145	0.9913
JPCLN058	0.9802	0.983	0.9656	0.9964

Table 3. Final model performance on three testing images. The table shows the segmentation performance of the final model (*model d* of Tab 1) on the three images shown in Fig 2.

different metrics evaluating the final model segmentation of the three images. We can see that DSC, accuracy and specificity agree that the model obtained the worst segmentation performance on the first image, while the worst sensitivity was obtained by the segmentation of the second image. Another method that could have been used to assess the model performance is to perform a receiver operating characteristic (ROC) analysis. In fact, a ROC curve shows the performance of a binary classifier, by comparing sensitivity versus specificity (1 - specificity) for every possible classification threshold. The area under the ROC curve (AUC) is a derived summary measure for the accuracy of the model.

3 Cross validation

Cross-validation (CV) or k-fold cross-validation is a common machine learning technique used to randomly split the training data into k groups. At each iteration of the CV algorithm, one group is used to test the model performance and the others are used as training data. The final performance is evaluated computing the average error across all iterations. This is especially useful to test model generalization performance when the dataset has a limited size. Alternatively, it can be used in combination with the hold-out test method to perform hyperparameters tuning. In fact, a common practice is to split the data in advance into train and test set. Thus, CV can be used to randomly generate a validation set from the training data, which is used to perform hyperparameters selection. Lastly, once the final model is developed, its generalization performance can be evaluated on the test set, which should be previously unseen data. There are different variants of the CV algorithm, such as leave-one-out CV (LOOCV), stratified CV, and nested CV. In LOOCV the number of folds k equal to N, which are the number of samples present in the data. In stratified CV, the splitting procedure ensures to obtain the same proportion of classes in each fold. In nested CV, a nested k-fold CV is performed within each fold of the cross-validation.