Final project for Deep Learning

Claudio Facchinetti - 223814
<claudio.facchinetti@studenti.unitn.it>
Alberto Bellumat - 213898
<alberto.bellumat@studenti.unitn.it>

1. Introduction

Note: the training and the evaluation on the whole test set are very computationally expensive for both tasks, so we included a trained version of the models and the predictions generated in the archive.
The purpose of this report is to provide indications about how the group faced the assignment given as a final project of the "Deep Learning" course at UniTN during the academic year 2020/2021, explaining the difficulties encountered, the technical solution proposed, and possible improvements.
The assignment requirements were to address two different machine learning problems using the notions of deep learning that we acquired during the course and the laboratory sessions; both problems were defined over a modified version of the Market-1501 dataset, which is a large-scale dataset for person re-identification.
The first task assigned is a classification problem and, starting from the CSV file provided, we had to develop and train a model able to predict several attributes for each image; in particular, we had to predict the following attributes for each image:

- gender;
- hair length;
- sleeve length;
- length of lower-body clothing;
- type of lower-body clothing;
- wearing hat;
- carrying backpack;
- carrying bag;
- carrying handbag;
- age;
- color of upper-body clothing;
- color of lower-body clothing.

The second task was the proper re-identification task: we had to return the images containing the same person for each input image.
Thus, two different models were developed for the two main tasks.
However, besides the two main tasks, we also considered the existence of junk images in the test set of the dataset; these kinds of images may have a negative impact on the accuracy of our models. Therefore, we also developed another model for identifying possible junk images in the test dataset.

2. Classification problem

To face the classification problem, we started by looking at the dataset and, in particular, the annotations file: we noticed that the upper and lower color attributes were described by multiple labels, one for each color.
From the assignment text, only one label was set to 1 for each category, meaning that if the upper color is "blue", then the "upperblue" category has to be set to one while all the others need to be set to zero.
This was a problem for us because we had to instruct our model that there is some correlation between specific labels, and it was unclear how to do it. To solve this, we compressed these kinds of labels into one general label; for example, "upcolor", which takes N values, one for each label it represents.

2.1. Model for classification

The model we build was based on the concept of Convolutional Neural Network, using convolutional layers as feature extractors and then many final Multi-Layer Perceptron architectures to predict the labels. In particular, the whole architecture is composed of two convolutional blocks and twelve MLP, each one having two dense layers.
The first convolutional block is composed of two convolutional pipelines: a batch normalization layer, an actual convolutional layer, a linear activation function (in this case,

we used LeakyReLU), and then a poling layer (in this case, we used a MaxPool).
The second convolutional block is almost identical. The only key difference is that we did not use the pooling layer after the activation. We decided to do this because we wanted to experiment with the residual connection on this block, such that the network can use an identity connection without explicitly learning it.
The output of the model is a dictionary having as key an identifier of the label and as value the corresponding predicted value; when creating the prediction file, we had to unpack the compressed labels to make it coherent with the CSV file containing the training annotations.
All the choices of learning hyper-parameters, loss function, and optimizer were made empirically by testing many different configurations. We decided to keep the one that resulted in better accuracy and better generalization performances.
Regarding the dataset, we wanted the model to work even on unseen images; therefore, we decided to proceed with a data augmentation procedure to create slightly modified images from the original ones in the training set and we trained the model on these generated samples. We also split our original training set into a validation set and a smaller training set, ensuring that all the images of a person either appear in the training set or the validation set.
For the learning, we implemented early stopping: after every pre-defined number of epochs, we check if the performance is increasing. If there is no improvement, we stop the training. If there is an improvement, then we store the model and continue the learning. By improvement, we mean when the validation loss decreases.

Model	Validation accuracy (%)
Without residual connection	70.24
With residual connection	79.87

Table 1. Accuracy of model with and without residual connection

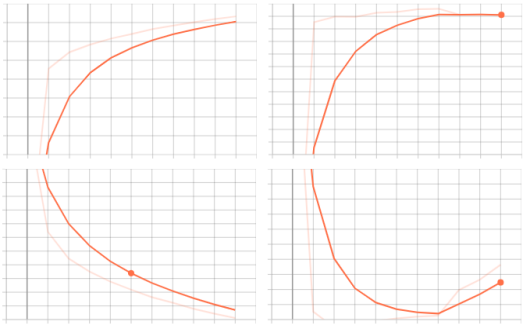

Figure 1. The top graphic represents the training and testing accuracy, the bottom one represents the training and testing loss during the training process.

2.2. Possible improvements

Instead of reinventing the wheel, it could be possible to use a pre-trained model, for example from the ResNet family, and then apply a transfer learning technique in order to have better performance, but we wanted to get our hands dirty trying to understand how things work and the actual impact.
As for the dataset, we could also use the Variational AutoEncoder we proposed for the second task to create new samples, but we would have to annotate them manually.

3. Person Re-Identification problem

To face the second task, we tried, in the first place, to develop a multi-label classification solution in which the possible labels were the people identities retrieved from the filename.
This approach could work if we have to pick the images from the training set, but this is not the case: we have to pick the images from the test set of the re-identification problem, and we do not have the identity information. This situation is quite challenging to manage because we cannot train our model on recognizing the identities of the images from which we have to pick the result of the query. Also, we need to consider the possibility of training the model on the training set of the first task, predict the identities of each image in the classification test set and compare these with the identity prediction of the query image, retrieving indeed the images belonging to the same person.
Theoretically, this method could work, but there is usually a massive abyss from theory to practice, especially if we do not have any proof of this intuition to be correct; therefore, we dropped this idea. In the end, the model we developed is essentially a slightly tweaked version of a plain Variational AutoEncoder which was used as a baseline to perform clustering.

3.1. Autoencoder for anomaly detection

In order to smooth the prediction process of our model for the second task, we decided to try and develop a convolutional AutoEncoder which could help us identify the junk images in the test set. After that, the junk images in the test set were deleted.
In the AutoEncoder, we designed for the encoder three convolutional blocks, whose internal structures are based on the VGG-19 network layers. At the end of each convolutional block, we apply a max-pooling layer, and we also recorded the locations of the maxima within each pooling region in variables; these variables were used later on for the unpooling operation.
We designed three deconvolutional blocks for the decoder part, whose structures were symmetric to the convolutional blocks and were based on the work presented by Zeiler and

Fergus[3]. At the beginning of each deconvolutional block, an unpooling layer was applied, using the variables stored in the encoder part.
We trained the AutoEncoder to learn a relevant latent representation, without the KL-Divergence, in order to make it able to detect anomalies, namely images that had a discrepancy with the others. For the training, we used data augmentation and early stopping.
After the training on the train set of the dataset, we used the whole AutoEncoder to reconstruct the images in the test set; given an input image, if the AutoEncoder output the generated image with a reconstruction loss equal or greater than a pre-defined threshold, then the input image was considered to be a possible junk image. Thus, at the end of the process, we had a list of possible junk images, such as, for example, images containing no people. As reconstruction loss, we used the mean squared error loss.
Because we were going to use the AutoEncoder for deleting data from the test set, we needed to be sure that we were deleting only junk images; we could not allow any mistake in the predictions generated by the AutoEncoder. So, given the list of possible junk images, we checked manually that the list contained only images that could be correctly considered junks. After the manual filtering, we deleted from the test set all the junk images. After the deletion process, we applied a Variational AutoEncoder in conjunction with a clustering algorithm (this is explained deeper in the next subsection).
The whole list of images that we identified as junks and the code needed to delete them from the test set are present in the source code of the second task.

3.2. Variational AutoEncoder

To address the second task, we wanted to exploit the fact that a Variational AutoEncoder (VAE) can learn an efficient latent representation of images in order to be able to reconstruct the original input from it. As we studied during the course, the Variational AutoEncoder loss function is composed of two different losses: the reconstruction loss and the KL-Divergence; this combination of losses is what allows the AutoEncoder to learn a latent space which is dense and in which all the samples of the same class are close in the space.
The dense latent space of a VAE was crucial for our solution because, in our context, we may consider a person as a class, meaning that the VAE learns a latent space in which all the images of the same person are close in the space. Therefore, we decided to propose and develop a solution that involved a VAE and a clustering algorithm.
First of all, we created the structure of a plain Variational AutoEncoder by also checking the original paper in which this approach was first introduced [2]. We tried many structures for both encoder and decoder, but we obtained unsatisfying results as the reconstructed inputs were blurry and grayscale.
Firstly, we realized that the grayscale problem was because, in order to be able to use the binary cross-entropy loss function, we appended a sigmoid activation function to the decoder. To avoid this, we then moved to the squared mean error loss function and substituted the sigmoid with a hyperbolic tangent, but the problem was not solved.
By inspecting deeper the raw values that the CNN was producing, we realized that the three channels were almost identical, and this what was causing the grayscale issue, so we tried to remove the final activation function solving the problem.
After this, we realized that the reconstructed images were not even close to the samples, resulting in random symmetrical shapes, and increasing the number of training epochs seemed to not work. To solve this problem, we tried to learn weights to allow a better reconstruction, resulting in us appending a final convolutional layer to the decoder so that the network could learn the filters to generate images closer to the original ones. This approach helped us to make the reconstructed images looking more like the original inputs.
We also had many problems handling rectangular images because the computation of the output size of the deconvolutional layers implies setting an appropriate size for the kernels, the strides and the output paddings; since we wanted to have the freedom of test out different architectures for both the encoder and the decoder network we decided to resize the input images to 64x64 pixels, keeping the shortest dimension of the original images.
All problems seemed to be solved, except for the fact that the reconstructed input was too blurry: we could not recognize any person-like shape into it. We tried to change the learning rate, employing an exponentially decaying logic, but nothing seemed to work. We realized that this could be due to several things:

- KL-Divergence: this was dominating the loss function, so we followed the advice of a paper [1] and tried to add a weight to it;

- Vanishing gradient: we realized that the MSE loss was an average and that the input image values were in the $[0, 1]$ interval; to solve this, we changed the MSE loss aggregation to sum, and we tried to double the values of the pixels of the original input, mapping them into the $[-1, 1]$ interval. This caused us to normalize back the image again when checking by hand the quality.

In the end, the quality of the reconstructed images was adequate: the images were still blurry, but we could clearly distinguish a person into them.

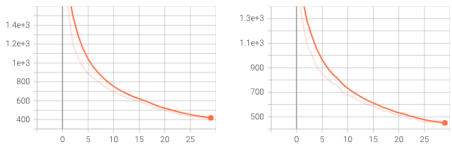

Figure 2. On the left is the training loss and on the right is validation loss respectively

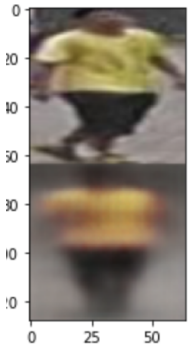

Figure 3. Example of reconstruction: on top the original one, on the bottom the reconstructed one.

3.3. Clustering the images

After the Variational AutoEncoder was trained, we could use it to predict the identity of each person in the query set. In particular, we first computed the latent representation of every image in the test folder, and then we clustered them using a hierarchical clustering function from the SciPy library.

For each cluster that we obtained, we computed its centroid as the mean vector of the latent representations of every image in that cluster; for each image in the query set, we obtained its latent representation via the Variational AutoEncoder and assigned it to the nearest cluster, basing our decision on the distance between the latent representation and the clusters' centroids.

As a base metric, we tried to compute the mean Average Precision metric (mAP for short) on the train set, and it resulted quite satisfying: the final accuracy was about around 60%.

The hyper-parameters used for both the training and the clustering were proven to be optimal only by empirical experiments: we tried several values and then realized which ones gave us the best performance.

3.4. Possible improvements

A step further would be to change the type of Variational AutoEncoder in use: instead of a plain VAE, we could use an InfoVAE, using the model for the previous task to provide also labels helping in the learning of a much expressive and robust latent space.

Another improvement could be the clustering algorithm: we tried to re-perform the cluster for every query image, but this was infeasible due to the computation time; we could try to change the clustering approach, maybe employing the Gaussian Mixture Model approach or any more sophisticated method. An alternative could be developing a model that also learns the thresh to maximize the mAP metric.

References

[1] Samuel R. Bowman, Luke Vilnis, Oriol Vinyals, Andrew M. Dai, Rafal Jozefowicz, and Samy Bengio. Generating sentences from a continuous space. *arXiv*, 2016. 3

[2] Diederik P. Kingma and Max Welling. Auto-encoding variational bayes. *arXiv*, 2014. 3

[3] Matthew D. Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. *arXiv*, 2013. 3