Homework3

Lorenzo Vaiani

5 January 2020

Abstract: in this study DANN technique is used to solve a domain adaptation problem on the "PACS" datast. Different combinations of learning rate, step size and alpha are used to train and evaluate a model based on AlexNet but that also implements DANN. The results with and without DANN are comapred in order to discover if this technique can make an improvement on this dataset.

1 Domain Adaptation

The Domain Adaptation (DA) problem consists in having inputs at test time that differ significantly from the training data used to train the model, which for this reason may not perform very well. In other words, the model does not work as expected even if the task (that is to assign the labels) remains the same because the domain of the input data changes. In particular, in the field of CNN it is a matter of treating images at test time which belongs to the same classes of the training labels but that differ from the input images in style, scenarios, acquisition system and other factors like these.

In the Unsupervised DA the source data is the only set of data of which the labels are known, while the target data is the set of data that contains images from another domain and without labels but which are still available at training time in order to perform techniques that can reduce the gap between the two domains. On the other hand, if some or all target images are labeled we are in semi supervised DA situation or supervised DA situation respectively.

There are several techniques to solve the DA problem. They can be distinguished according to the type and the number of source domains managed (figure 1):

- Single, if we have only one source domain.
- Multi, if there is more than one source and we know that these sources belong to different domains, so each image has not only its class label but also a domain label.
- Mixed, if there is more than one source but we do not know the domain label of the images.

Another subdivision can be made regarding the correspondence between the class labels target data and those of source data (figure 2):

- Open Set DA, if all the target images belongs to the same categories of the source images.
- Partial DA, if in the source data there are images that belong to a class which is not present in the target data.
- Closed Set DA, if both source data and target data have images that belong to a class which is not present in the other set of data.

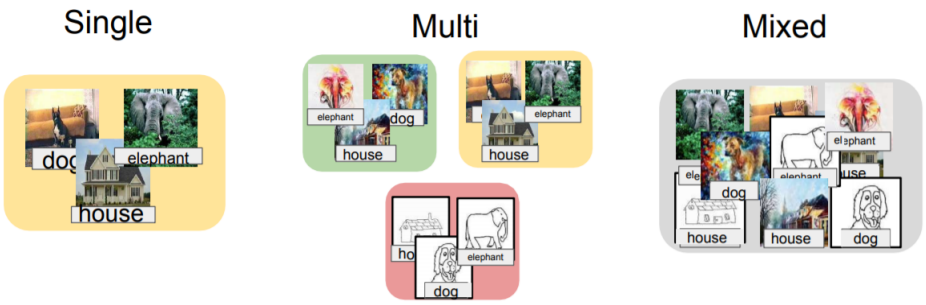

Figure 1: Different type of domain sources.

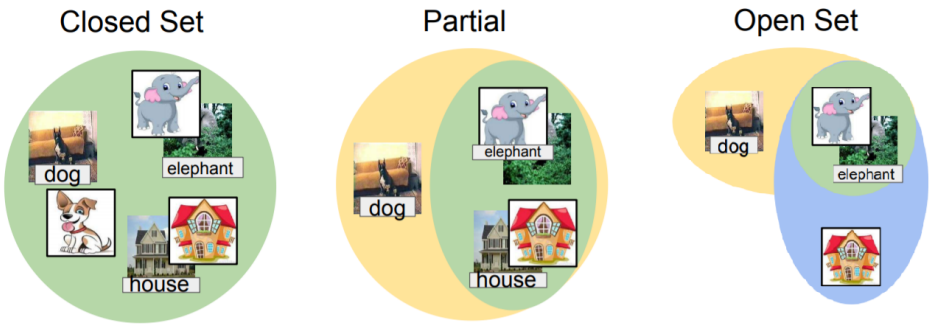

Figure 2: Different type of label correspondence.

The DA technique used in this study is called Domain-Adversarial Neural Network (DANN). The basic idea is that to get a model that can generalize well from one domain to another, the internal representation of the neural network has to contain no discriminative information about the origin domain of the input, while preserving a low risk on the source samples of which the class label is known. This technique aims to align the source data distribution with the target data distribution, so that knowledge learned from the source can be used to obtain good predictions also on the target data.

A possible architecture able to implement this technique is based on a classical CNN architecture (convolutional layers as feature extractors plus fully connected layers as label predictors) to which a domain classifier is added (figure 3). This classifier is connected to the feature extractor via a gradient reversal layer (GRL) that multiplies the gradient by a certain negative constant during the backpropagation-based training. There is no parameters associated with the GRL. It acts as an identity transformation during the forward propagation, while it takes the gradient from the subsequent level and changes its sign before passing it to the preceding layer during the backward propagation. In this way, the neural network and the domain classifier are competing against each other, in an adversarial way, because the parameters of the network are updated in order to minimize the loss of the label predictor and to maximize the loss of the domain classifier at the same time.

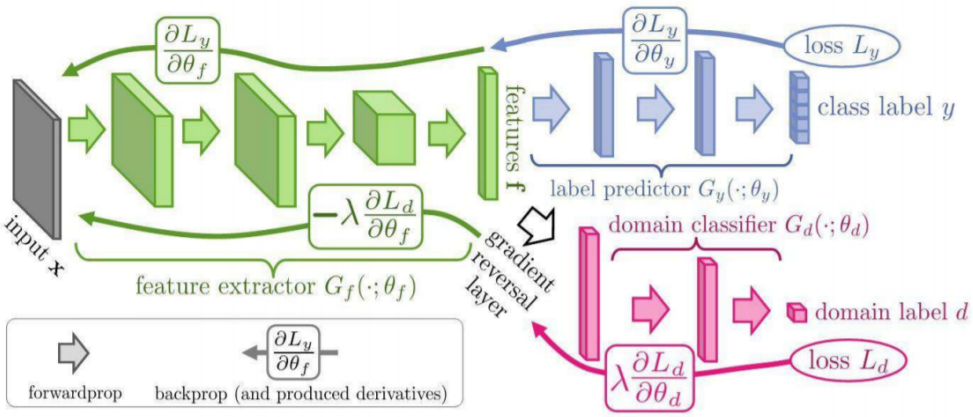

Figure 3: DANN architecture scheme. The green layers are the feature extractor, the blue layers are the the class label predictor and the red layers are the domain predictor.

If we consider the mathematical details, we can defined the label prediction loss and the domain prediction loss respectively:

$$L_y^i(\theta_f, \theta_y) = L_y(G_y(G_f(x_i; \theta_f); \theta_y), y_i),$$

$$L_d^i(\theta_f, \theta_d) = L_d(G_d(G_f(x_i; \theta_f); \theta_d), d_i).$$

where $G_f(\cdot;\theta_f)$ represents the feature extractor part of the network with parameter θ_f, $G_y(\cdot;\theta_y)$ represents the class label predictor with parameter θ_y and $G_d(\cdot;\theta_d)$ represents the domain predictor with parameter θ_d. So training DANN consist in minimizing:

$$E(\theta_f, \theta_y, \theta_d) = \frac{1}{n}\sum_{i=1}^{n} L_y^i(\theta_f, \theta_y) - \lambda\left(\frac{1}{n}\sum_{i=1}^{n} L_d^i(\theta_f, \theta_d) + \frac{1}{n'}\sum_{i=n+1}^{N} L_d^i(\theta_f, \theta_d)\right)$$

by finding the saddle points $\hat{\theta}_f, \hat{\theta}_y, \hat{\theta}_d$ such that:

$$(\hat{\theta}_f, \hat{\theta}_y) = \underset{\theta_f, \theta_y}{\operatorname{argmin}} E(\theta_f, \theta_y, \hat{\theta}_d)$$
$$\hat{\theta}_d = \underset{\theta_d}{\operatorname{argmax}} E(\hat{\theta}_f, \hat{\theta}_y, \theta_d)$$

These saddle points can be obtained calculating the stationary points of the following gradient updates:

$$\theta_f \longleftarrow \theta_f - \mu \left(\frac{\partial L_y^i}{\partial \theta_f} - \lambda \frac{\partial L_d^i}{\partial \theta_f} \right),$$

$$\theta_y \longleftarrow \theta_y - \mu \frac{\partial L_y^i}{\partial \theta_y},$$

$$\theta_d \longleftarrow \theta_d - \mu \lambda \frac{\partial L_d^i}{\partial \theta_d},$$

where μ is the learning rate. These updates are very similar to stochastic gradient descent (SGD) updates for a feed-forward deep model that has a feature extractor part linked to a class label predictor and to a domain label predictor (with loss multiplied by a factor called λ). The only difference is that in the first of these updates the two gradients are subtracted, instead of being summed. This effect can be obtained introducing the GRL mentioned above, which allows us to use SGD to update the gradient in our network implementation.

2 Network and Dataset

In this study a version of AlexNet CNN that implements DANN is realized. To do this the code of AlexNet (at this link) has been modified in order to add some layers and to change the forward function. The convolutional layers and the fully connected layers are the same as the original version, except for the last fully connect layer that is adapted to the number of output classes. As a DANN domain classifier, a branch of the network, parallel and identical to the fully connected layers, has been added. The last layer of this branch has only to neurons used to distinguish between the target domain and the source domain. The forward function has been modified so that it can decide to perform one of these two different operations:

- Let the images follow the normal path in the network, through the convolutional layers and the old fully connected layers in order predict the class label. This concerns only the images from the source domain.
- Let the images follow the alternative path, through the convolutional layers and the new branch of fully connected layers in order predict the domain label. This concerns the images from both the source and the target domains.

The dataset used in this study is the PACS dataset (at this link) that contains images that belong to four different domains (Photo, Art Painting, Cartoon and Sketches) and represent seven different classes. The images in photo domain (1670 images) are used as source set, the images in art painting domain (2048 images) as target set and the images in cartoon domain and sketch domain (2344 and 3929 images respectively) as validation set.

The images are resized and cropped in the center in order to adapt them to the size requested in input by AlexNet. Furthermore, since the pretrained version of AlexNet is used, the images are normalized using particular mean and standard deviation values and the weights of the original fully connected layers are copied in the new domain classifier layers.

3 Training without DANN

The function "training_phase()" trains the network without ever considering the DANN branch layers. This function train the network several time, testing different combinations of learning rate and step size for 30 consecutive epochs. At each epoch of each hyperparameters combination the network calculates the accuracies on the two validation sets and averages the results obtained. If this final accuracy is the highest found so far, the model that produced it is stored to be returned at the end of the function.

This model is taken as input from the function "test_phase()", which uses it to predict test sample labels. At the end of this function the predicted labels are used to calculate the accuracy on the target domain set.

The chosen loss is the Cross Entropy Loss function, the chosen optimizer is the SGD optimizer and the chosen scheduler is the Step Low scheduler, all implemented by Pytorch. The values used to built the combination of hyper parameters are:

- Learning rate $\in$ [0.01, 0.005, 0.001]. Higher values of learning rate make the loss reach the NaN value.
- Step size $\in$ [10, 20].

The other hyperparameters are fixed, such as batch size equals to 256, momentum equals to 0.9, weight decay equals to $5 * 10^{-5}$ and γ equals to 0.1.

In Table 1 it is possible to see the average accuracy calculated with the accuracies on the cartoon domain set and sketch domain set. Moreover for each accuracy it is indicates at which epoch it was obtained. Noteworthy the fact that the best results are obtained with the higher value of the learning rate. Another interesting fact is that often the best accuracy for a certain combination of hyperparameters is reached within the first ten epochs.

The best result is obtained with an initial learning rate of 0.01 and a step size of 10. Using the model that produce this result to calculate the accuracy exploiting the target domain set, the final value is an accuracy of 49.36%. This accuracy is quite higher than those obtained at validation time on cartoon and

sketch datasets: this is probably happens because the art painting images, that make up the target domain set, are much more similar to photo images than cartoon and sketch images are.

	LR = 0.01	LR = 0.005	LR = 0.001
SS = 10	35.72% Epoch = 7	29.75% Epoch = 10	28.93% Epoch = 28
SS = 20	33.13% Epoch = 13	30.75% Epoch = 4	26.90% Epoch = 1

Table 1: average validation accuracies for all the combinations of hyperparameters training the network without DANN.

4 Training with DANN

The function "DANN_training_phase()" trains the network exploiting also the domain classifier layers. As for the training without DANN, this function tests several combinations of learning rate, step size for 30 consecutive epochs. Moreover, the search to find the best value is also carried out for hyperparameter alpha (that is the factor called λ in the first section, but the word "lambda" is inconvenient to use in python because it is a keyword).

At every run this function uses only one of the two validation domain set available. At each execution the images in the source domain pass through both the label classifier and the domain classifier of the network, while the images in the validation domain set go through only the domain classification branch. Of course, when the images pass through the domain classifier their real label is removed and another one that represents the domain to which they belong to is assigned. These three steps are performed in parallel batch by batch. Next, the computed gradient, updated from all these three operations, is used to update the weights of the network. The function is capable of handling situations in which the source set and the validation set have a different quantity of batches, so in the last iterations of each epoch it is possible that the gradient is updated only by some of the three steps mentioned above. Finally, at the end of each epoch, the same validation domain set used for training is used to calculate the accuracy which will be used to identify the best combination of hyperparameters. The function returns the list of accuracies at each epoch for each hyperparameter combination. This function is called twice, once for each validation domain dataset.

Then, the corresponding accuracy of each call are averaged and the highest result thus obtained will indicate the best combination of hyperparameters. Unlike training without DANN, in this case only the accuracies obtained in an epoch greater than the third one are considered. This is done in order to increase the stability of the results because sometimes a certain combination of hyperparameters reaches its highest accuracy in the first three epochs due

to the randomness of some factors, such as the random composition of the batches or the random weights in the replaced output layers. Thereafter, if that combination is chosen as the best one, that number of epochs should be reused to generate the final model using target domain images, but training it for so few epochs would result in poor testing results (because the initial fortuitous randomness that characterized the validation phase is unlikely to happen again).

The function "DANN_final_training()" use the hyperparameters values highlighted by the previous function to create the model that will be used to calculate the accuracy on the target domain set. It is practically identical to the previous function, but there is no validation step where the accuracy is calculated and only one model is trained and returned by the function. This model is taken as input from the function "test_phase()" that computes the final accuracy on the target domain set, as for training without DANN.

The values used to built the combination of hyper parameters are:

- Learning rate $\in$ [0.001, 0.0005, 0.0001]. These values are an order of magnitude lower than those used in the experiments without than, again because higher learning rates make the loss reach the NaN value.
- Step size $\in$ [10, 20].
- Alpha $\in$ [0.5, 0.3, 0.1].

The loss function, the optimizer, the scheduler and the other hyperparameters are the same used for training without DANN.

In the tables 2, 3 and 4 it is possible to see the average accuracy calculated with the accuracies on the cartoon domain set and sketch domain set, divided by the alpha value used. Moreover for each accuracy it is indicates at which epoch it was obtained.

	LR = 0.001	LR = 0.0005	LR = 0.0001
SS = 10	26.25% Epoch = 10	23.30% Epoch = 14	21.35% Epoch = 8
SS = 20	24.23% Epoch = 14	30.62% Epoch = 20	30.28% Epoch = 6

Table 2: average validation accuracies for all the combinations of hyperparameters training the network with DANN, using an alpha value equals to 0.5.

It is noticeable that, in general, a higher step size allows the network to achieve a better result. Compared to the results obtained during the validation phase without using the DANN technique, these ones are slightly lower. The best result is obtained with a step size of 20 and an initial learning rate of 0.0005, which is not the highest among those tested. Training a model with the hyperparameters that produce this result and calculating the accuracy exploiting the target domain set, the final value is an accuracy of 50.53%.

	LR = 0.001	LR = 0.0005	LR = 0.0001
SS = 10	21.86% Epoch = 7	21.20% Epoch = 21	21.71% Epoch = 6
SS = 20	28.36% Epoch = 14	26.68% Epoch = 18	19.10% Epoch = 24

Table 3: average validation accuracies for all the combinations of hyperparameters training the network with DANN, using an alpha value equals to 0.3.

	LR = 0.001	LR = 0.0005	LR = 0.0001
SS = 10	29.33% Epoch = 8	26.10% Epoch = 10	21.21% Epoch = 7
SS = 20	25.44% Epoch = 8	27.61% Epoch = 21	24.40% Epoch = 29

Table 4: average validation accuracies for all the combinations of hyperparameters training the network with DANN, using an alpha value equals to 0.1.

This accuracy, as in the previous case, is much higher than those obtained during the validation phase (for the same reasons) and is also slightly higher than that obtained without DANN. The low increase could be due to the fact that PACS dataset does not contain a sufficient number of samples to train a model that is quite relevant. This problem can be solved by starting with a model pretrained on e better dataset or obtained from another network architecture (such as VGG) and by using some data augmentation techniques. The poor improvement may also be due to the fact that the best combination of hyperparameters was found using two validation domain sets with a very different data distribution from the target domain set.