Deep Optimal Transport for Domain Adaptation Application to Vehicle Re-Identification

Yujin CHO
Master Student E3A
Université Paris-Saclay
Paris, France
amycho92@naver.com

Israfel SALAZAR
Master Student E3A
Université Paris-Saclay
Paris, France
israfel.sr@gmail.com

***Abstract*—Vehicle re-Identification aims to identify the same vehicle across the different cameras. It has useful applications in surveillance and intelligent transport systems. One of the fundamental challenges of vehicle re-identification is how to learn robust and discriminative visual features given in small inter-class similarity and large intra-class differences that don't follow the same distribution. In this project we propose to treat the re-identification problem as a domain adaptation task. We implemented the Deep Joint Domain Adaptation algorithm to train a model that could robustly detect the same classes even when the images were obtained in different conditions.**

***Index Terms*—Vehicle Re-Identification, Convolutional Neural Networks, Optimal Transportation**

I. Introduction

This paper presents our work implementing a vehicle re-identification (vehicle Re-ID) model using optimal transport techniques. Vehicle re-ID is the task of finding a queried vehicle from non-overlapping cameras comparing the attributes of the queried image and a data set of other cameras selecting the ones that present the same instance. Vehicle re-ID is part of the smart city concept that aims to improve the quality of life of city dwellers by making the city more adaptive and efficient, using new technologies that are based on an ecosystem of objects and services.

The advances in artificial intelligence, especially the deep learning techniques applied to computer vision, allowed us to implement models capable of outstanding results in object recognition and detection. However, the actual computer vision models still depend strongly on the data acquisition variables such as illumination, contrast or the device used to it. The Vehicle re-ID task must deal with a large intra-class difference in which same labeled images could have different distribution depending on the environment and angle of cameras. At the same time, Vehicle re-ID must face a large inter-class, where instances of different classes present high similarities such as color, shape and aspect. To solve these problems we implemented an optimal domain transport algorithm that could learn jointly the distribution of a source and target dataset and the classification task.

Optimal Transport [1] problem allows us to compare probability distributions in a geometrically manner. We search for the most economical way, for example in distance, to transport the distribution of a set of data to another using the geometry of the feature space. Deep Joint Domain Adaptation [2] (DeepJDOT) presents an algorithm to find this transformation maintaining the discriminant information of a classifier neural network. In this project we will train a Convolutional Neural Network to solve the Re-Identification Problem. For that, we trained a model in a Source dataset, then using the DeepJDOT algorithm we trained the model in a unsupervised manner over a different distributed Target dataset, extending the learned classification capabilities to this unknown data distribution.

II. Related works

A. Joint Domain for Optimal Transport

Optimal Transport [1] is a technique that allows us to compare probability distributions. It searches for a transport plan or joint probability distribution represented by a coupling $\gamma \in \Pi(\mu_1, \mu_2)$ which yields a minimal displacement cost. Equation 1 shows how to obtain the coupling matrix γ.

$$\gamma_0 = \underset{\gamma \in \Pi(P_s, P_t)}{\operatorname{argmin}} \int \mathrm{d}(x_1, x_2) d\gamma(x_1, x_2) \tag{1}$$

Here $\Pi(\mu_1, \mu_2)$ describes the joint probability space with marginals probabilities μ_1 and μ_2.

Now let's define some notation, we will denote $X_s = \{x_i^s\}_{i=1}^{N_s}$ the training set in which every item is corresponded with a element in $Y_s = \{y_i^s\}_{i=1}^{N_s}$ the label set. In a common learning setup we will also have a test set $X_t = \{x_i^t\}_{i=1}^{N_t}$ with unknown labels. To determine Y_t, the general approach tries to find the joint distribution $P(X, Y)$ and assume that X_s and X_t have the same probability distribution. Joint Domain for Optimal Transport tries to tackle the problem when this assumption no longer correct, ie. there exist two different probability distributions $P_s(X, Y)$ and $P_t(X, Y)$. For simplicity, we will call P_s and P_t the marginal probabilities $P(X_s)$ and $P(X_t)$.

Courty et al [3] proposed to find a solution to the change in both, the marginal and the conditional distributions finding a map $\mathcal{T}$ that aligns directly P_s with P_t. Following the Optimal Transport Plan they propose to minimise the Equation 2.

$$\gamma_0 = \underset{\gamma \in \Pi(P_s, P_t)}{\operatorname{argmax}} \int \mathcal{D}(x_1, y_1; x_2, y_2) d\gamma(x_1, y_1; x_2, y_2) \tag{2}$$

Where $\mathcal{D}(x_1, y_1; x_2, y_2) = \alpha \mathrm{d}(x_1, x_2) + \mathcal{L}(y_1, y_2)$ is a joint cost function measuring the distance between the features of the samples plus a loss function that measures the discrepancy between the labels. We can see that differently to the approach of Optimal Transportation(Equation 1) that only searches to align the latent space, here we are also jointly learning the classifier adding it in the loss function.

Equation 2 assumes a label for both the source and the target domain. When applying it to unsupervised learning, we don't have the target labels. In [3], they proposed to work with a classification function $f(x)$ instead of y_2 defining a new joint probability $P_t^f(X, f(x))_{X \sim \mu_t}$.

The problem is then reduced to find the function f for the target domain. To do so, we can convert the minimization above (Equation 2) into a discrete problem, given that the distributions with which we are dealing are discrete. In this case γ is a matrix in Δ, a transformation between uniform distributions. The authors proposed the new minimization problem.

$$\min_{f,\gamma} \sum_{ij} \mathcal{D}(x_1, y_1; x_2, y_2)\gamma_{ij} \tag{3}$$

In practice, a regularization term on f will be added to the function above to avoid overfitting in the classifier. In the paper, the authors proved that as a byproduct of this minimisation, closer samples will be matched with similar labels. This solution has two important drawbacks: the first is that the search for the matrix γ scales quadratically with the increase in the amount of data, making it impossible to work with large datasets. The second is that the differences between the source and target sets are computed at the input level, so there are no features to compare at that level.

B. Deep Joint Domain for Optimal Transport

Deep Joint Domain for Optimal Transport (DeepJDOT) [2] addresses the drawbacks of the previous method using stochastical gradient using small batches of data to compute the matrix γ and performing the comparison between the samples in the feature space. It consist of two functions $g : \mathbf{x} \rightarrow \mathbf{z}$ and $f : \mathbf{z} \rightarrow \mathbf{y}$. The first one is an embedding function that takes an input and transforms it into the latent space of features $\mathcal{Z}$. The second one is a classifier that given the latent space output a label on the source and target domain. As in JDOT, both functions are jointly optimized to perform well on the target domain as shown in Figure 1.

C. Re-Identification task

Deep learning based Re-identification (ReID) is widely studied in the field of computer vision with main implementations for Person re-ID [4] and Vehicle re-ID [5]. Recently, both tasks have progressed but the problem of Vehicle Re-ID could be more challenging due to the most attention that has been paid to person re-identification in recent years due to the abundance of well-annotated pedestrian data, along with the historical focus of computer vision research on human faces and bodies. Moreover and as mentioned before, the difficulties due to the small inter-class similarity and large intra-class make the task a difficult computer vision problem.

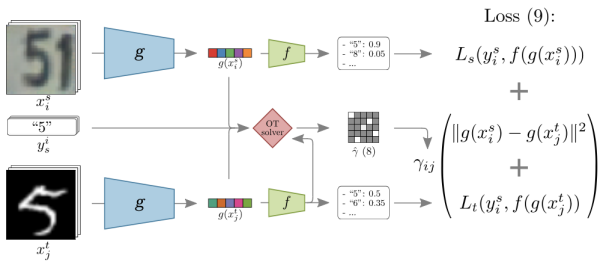

Fig. 1. Overview of the DeepJDOT method used.

III. Deep Domain adaptation for Vehicle Re-ID

Actual machine learning models need several data to be trained properly and to create datasets big enough is expensive and time consuming. Transfer learning came as a solution to these difficulties allowing us to re-use a model which is already trained on a very large datasets. As a result, even with a relatively small number of data, we can train deep learning models that learn its distribution.

Domain adaptation is a kind of transfer learning in which we have two different datasets, source and target, both with different distribution on the samples but the same overall classes. The learning can be performed in a supervised or unsupervised or semi-supervised manner.

In this work we implemented a Vehicle Re-ID method able to identify the same vehicle across different cameras. Some of the problems that we faced are:

- The position of the camera leads to different view angles. Vehicles have a variety of shapes (SUV,bus,sedan,pick-up,...) that translate into big differences on the images of a class depending on angle.
- The environment has a big impact as it will make distribution of the images to shift due to external impacts. For instance, obstacles, illumination, blurred pictures, among others.

Our hypothesis was that using semi-supervised domain adaptation can overcome these difficulties. For that, given two dataset with different distribution, source and target, we can summarize our implementation into the following points:

- We trained our model using the source dataset, so it performs well on the source test.
- We tested the performance of the model in the target dataset.
- We adapted our model to the distribution of the target using the domain adaptation method DeepJDOT.
- We tested the new performance on both dataset achieving a better performance on the target dataset and maintaining the performance on the source.

IV. Implementation

A. Dataset Preparation

In this work, we used the VeRi-776 dataset [6] that is a public vehicle dataset provided by the Beijing University of Posts and Telecommunications on ECCV built from the VeRi dataset. It was collected with 20 different real-world traffic surveillance cameras and contains a total 775 vehicles; 200 vehicles for testing and 576 vehicles for training. Figure 2 shows a sample of the dataset. Each car was captured by 2 to 18 cameras with different angles, resolution and illumination reflecting the real conditions of re-ID tasks. The dataset also contains bounding boxes, types, colors and brands and there several vehicles include license plates helping to the re-Identification.

Fig. 2. Example of VeRi-776 dataset

For our purpose we had to divide the dataset into a source and target subset. Both needed to have distinct distribution in order to simulate the environment to the DeepJDOT method. To do so, we analysed the images and and the different camera's distributions. We splitted the data into four main categories that are shown in the Figures 3, 4, 5 and 6.

Fig. 3. Set 1: Front and Rear View

Fig. 4. Set 2: Side View. Images in which the vehicles are shown on their left and right side.

The Table I shows the match of the defined sets and the cameras with more instances in the specific set. Since we wanted to have two different distributions we decided to use the Source Dataset, the Front/Rear set (Figure 3) and all the other sets as the Target Dataset.

Fig. 5. Set 3: $3/4$th View, Images showing the side view plus the front or the back of the vehicles.

Fig. 6. Set 4: Dark Environment and/or Obstacle Disturbing the Vehicles

B. Dataloaders and Preprocessing

After selecting the data to form the source and the target dataset we needed to prepare the data so we could input it to a model. To do so, we used in the whole implementation the PyTorch deep learning library. Pytorch includes different tools to import and process the data. Particularly, in this project we used the classes Dataset and Dataloader to implement an iterator on the datasets and the class transform to preprocess the images.

To build the Dataset object we read the file containing the information of all the vehicles. This file contains rows with comma separated values in which each row has the information of the file name, the label, the camera and other relevant data of an image. To create the Dataset we transformed the file into a Dataframe using the Pandas library to perform a more easy manipulation of the columns and rows of the data then we selected the file name, the camera and the label and dropped the other categories.

Once the Dataframe was ready, we designed the Dataset object to fulfill two important conditions: First, we wanted to select specific cameras to create the source and the target. Second, to obtain good results on DeepJDOT we needed that the classes inside each target dataset (train, validation and test) were a subset of the classes present in the source datasets. For that we created complementary functions that filter the data according to the classes and the cameras returning only the valid Dataset.

After the Dataset, the pipeline of the data was quite straightforward. We implemented three different transformations. The first was a resize, the model needs all the input data in the same size. Then, we transform the image file into a tensor, here it is important to recall that the image representation in PyTorch is $CxHxW$ instead of the more classic $HxWxC$. Finally, we implemented a normalization of the images.

C. Model Selection

The choice of the model is an important part of the project that could dramatically affect the obtained results. As

TABLE I
DIVISION OF SOURCE AND TARGET CAMERAS

source camera
CAM 01
CAM 02
CAM 06
CAM 10
CAM 11
CAM 13
CAM 14
CAM 15
CAM 17
CAM 19

target camera	information
CAM 03	Side view
CAM 04	Side view
CAM 05	Blurred image
CAM 07	Side view
CAM 08	Dark environment and tree
CAM 09	Dark environment and tree
CAM 12	3/4 view
CAM 16	3/4 view

described previously (see Figure 1), the DeepJDOT method needs to access to some specific layer of the model, that's why we decided to use two different models:

1) Feature extraction model: We used a convolutional neural network to extract the attributes of input images and reduce the amount of information into an embedded vector of the features.
2) Classifier: Using the embedded features's output as input this model computes a classification into the 775 different possible classes.

Given the characteristics of our problem, we decided to implement a ResNet50 which is a quite recent and versatil model that is presented in Figure 7. It is a variant of ResNet [7] model which has 48 Convolution layers along with 1 MaxPool and 1 Average Pool layer. This model is known to be deep and optimized in gradient propagation thanks to the residual blocks, it makes this model really efficient for machine vision activity. We decided to use a pretrained model with ImageNet that Pytorch has at disposal saving time during the training phases given that the model already knows which kind of features needs to be extracted to a good classification.

For the classifier model we simply added just 2 fully connected (FC) layers. The first FC layer has the same dimension as the embedded vector of features and the last FC layer output dimension corresponds to the amount of classes. A ReLU layer is used as the final activation layer.

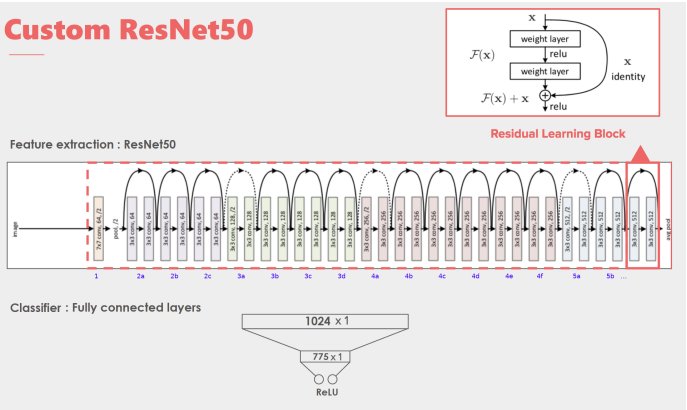

Fig. 7. Architecture of CNN based on ResNet50

D. Loss Functions

The core of the DeepJDOT algorithm rests on the implemented loss functions that allows the model to learn the distribution mapping and the classifier at the same time. The original paper [2] presented the functions using the Keras library so we decided to fully implement them using the PyTorch modules. For that we created two different classes that inherit from torch.nn.Modules. These classes need an input and a forward pass and they allow PyTorch to keep track of the gradients when computing operations with the vectors involved in the backpropagation step.

1) Gamma Loss: The implementation of the gamma loss follows the Equation (4) in which when the feature extractor model $\hat{g}$ and the classifier model $\hat{f}$ are fixed is reduced to a standard OT problem in which we can compute $C_{ij} = (\alpha||\hat{g}(x_i^s) - \hat{g}(x_j^t)||^2 + \lambda_t L_t(y_i^s, \hat{f}(\hat{g}(x_j^t))))$. Using this equation, we computed the first part of it using the euclidean distance in the feature space between the source and target images in the batch. Then, using the classifier model get the prediction labels for the target data and perform a cross entropy to compute C_{ij} Finally, using the POT library we computed the Wasserstein distance between the samples that solves the OT problem obtaining in this way gamma. The final loss function in the forward pass uses gamma to compute the difference between the feature spaces of the source and the target after mapping them with the transformation matrix gamma.

$$\min_{\gamma \in \Pi(\mu_s, \mu_t)} \sum_{i,j=1}^{m} \gamma_{i,j}(\alpha||\hat{g}(x_i^s) - \hat{g}(x_j^t)||^2 + \lambda_t L_t(y_i^s, \hat{f}(\hat{g}(x_j^t)))) \tag{4}$$

2) Classifier Loss: The classifier loss function is simpler after computing gamma. It follows the Equation 5 and it's objective is to train the classifier model in the source as well as in the target. To do so, it gets as arguments the real labels of the sources and the source predictions, the predictions on the target and the gamma matrix. We performed cross entropy loss using both the source predicted labels and their original labels and between the target predicted labels and the original source labels mapped by gamma.

$$\frac{1}{m}\sum_{i=1}^{m} L_t(y_i^s, \hat{f}(\hat{g}(x_i^s)))) + \gamma \in \Pi(\mu_s, \mu_t) \sum_{i,j=1}^{m} \gamma_{i,j}(\alpha||\hat{g}(x_i^s) - \hat{g}(x_j^t)||^2 + \lambda_t L_t(y_i^s, \hat{f}(\hat{g}(x_j^t)))) \tag{5}$$

E. Training

The training was divided in two phases:

- Model trained on the source dataset using cross entropy loss and a common supervised learning technique.
- Model trained on the source and the target dataset using the semi-supervised DeepJDOT method.

1) Model Trained in Source Dataset: As mentioned before in the model selection section we start with a pretrained model in the ImageNET. During this first stage we performed a finetuning of our model, feature_extractor + classifier, to have a good performance over the source dataset. We trained for 15 epochs using a learning rate of 0.001. We used Adam Optimizer and we achieved a 0.94% of accuracy on the validation set.

$$\text{Accuracy} = \frac{\text{Number of correct answers}}{\text{Number of queries}} \tag{6}$$

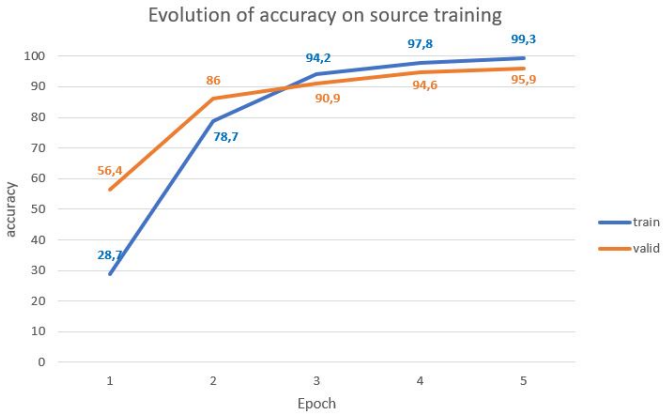

Fig. 8. source training accuracy

This metric gives us a good overview of how the model learned. During the test phase of our model we had an accuracy of 95.9% on source dataset in 5 epochs.

We can deduce that the ResNet model handled well the Vehicle Re-ID activity on source dataset.

2) Source and Target DeepJDOT Training: . The training function of the DeepJDOT implementation was a little more complex and it follows the Algorithm 1. At each forward pass we needed to call a batch of each the source and the target. Here we implemented a special iterator so at each time we call the source, we also receive a target batch in an endless manner.

Then, following the algorithm, we use the feature extractor and the classifier to compute the gamma matrix. We then use it to compute the gamma loss. The next step is to perform the update on the two models using also the gamma matrix and compute the classification loss. The last step is to do backpropagation.

Algorithm 1 DeepJDOT Stochastic Optimization

Require $\mathbf{x}^s$: source domain samples, $\mathbf{x}^t$: target domain samples, $\mathbf{y}^s$: source domain labels.
1: **for** each batch of source $(\mathbf{x_b}^s, \mathbf{y_b}^s)$ and target $(\mathbf{x_b}^t)$ **do**:
2: fix $\hat{g}$ and $\hat{f}$, solve for γ using Equation 4.
3: fix γ, and update for $\hat{g}$ and $\hat{f}$ according to 5.
4: **end for**

F. Evaluation

In this section we will approach the evaluation part of our project. First we will talk about the evaluation of the model on the data source part, it is a supervised training so we have access to the data label. We can therefore evaluate our model with the metric accuracy which corresponds to Equation 6.

In a second part, the evaluation of the model after its JDOT training, that we have made the model evolve to correspond to the distribution of target data. The labels of the target data are unknown to the model. To measure our model, we will use Ranking, by measuring similarities between the classes, we will be able to display the results which have the most similarities among a gallery of vehicles and the queries requests. The realization of this algorithm includes several stages as below :

1) First, we will have to compute the output value of our model's feature extractor for all vehicles in the gallery
2) Secondly, we will retrieve the value output value of our model's feature extractor for the query
3) Finally we compute the euclidean distance between all each score of the gallery array and with our query.
4) We will have an array of score and goal is to select the k-th first.

We can see above how the ResNet model react when we submit him a vehicle image from target dataset, even if he never seen this angle, the model has learned enought to extract the good features and identify really similar vehicles.

V. Results

A. Supervised model results.

Before performing evaluation using queries we performed the classic accuracy computation using the labels for the target and the source. Here it's important to mention that the accuracy on the target data is just a reference given that in the real problem we don't have the labels of the target dataset.

The Table II shows the accuracy reached for the model trained in the supervised fashion over the source dataset. We can see that the model was able to generalize the source data performing well on all the source datasets. This result was expected given that the Feature Extractor was pretrained on a dataset with 1000 classes and thousands of examples. On the other hand, we can also see on the table the drop of performance between the source and target and we can notice here the importance of the assumption that the datasets should be of the same distribution.

TABLE II
Accuracy for the Source Data Trained Model

	Train	Validation	Test
Source	0.896	0.94	0.8917
Target	0.3135	0.3291	0.3315

When we perform queries calls using the supervised source model, again and as expected we get good results for the source test dataset but the model is not robust enough to learn

the distribution of the target. Figure 9 shows some results for a query from the source dataset and tested on the target dataset for the supervised-trained model.

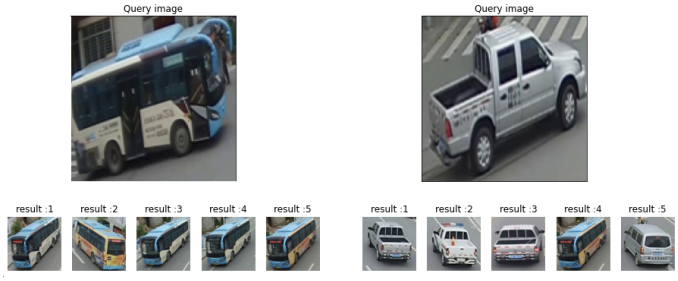

Fig. 9. Result of target query request on source data trained model

B. DeepJDOT model results.

When using the model trained using DeepJDOT we can see the results shown in the Table III for the accuracy on each dataset. We can see that the model was robust on the source dataset but it didn't generalize for the target. The DeepJDOT model even *forgot* some labels in the source and target datasets.

TABLE III
ACCURACY FOR THE SOURCE DATA TRAINED MODEL

	Train	Validation	Test
Source	0.8864	0.8942	0.8917
Target	0.1859	0.1807	0.1903

When performing query calls the results are the ones shown in Figure 10 and 11. Looking at the results, it's possible to conclude that the model learned some features that identifies the query vehicle, specially in Figure 11 where all the vehicles seems similar even to the human eye, but it mistakes the labels not selecting only the ones with the same class.

Fig. 10. Result of target query request on source data trained model

VI. ANALYSIS

As it's possible to observe from the results, the model trained with the DeeoJDOT algorithm wasn't able to correctly map the distribution of the target dataset to the one of the source dataset. In this section we present some analysis of the possibilities leading to these results.

Fig. 11. Result of target query request on DeepJDOT trained model

A. Dataset Complexity

One of the possibilities is that the dataset was too complex for the algorithm. On the original paper and code they trained the model using the MNIST dataset [8] in which the task is to identify handwritten digits with only 10 output classes with a high inter-class differences and high intra-class similarity exactly the opposite as our dataset in which the vehicle classes shared a big amount of attributes.

Even more, we can see from the results that the feature extractor wasn't the problem (the selected resnet model), because it was able to obtain good performance on the source dataset when it was trained in the supervised fashion and even in some experiments it obtained better results on the target dataset than the model trained with the deepjdot algorithm.

The algorithm even had problems to overfit a small dataset. We tried testing the algorithm selecting only a few images so the model could overfit on the source and the target but the algorithm could learn by heart the images as supposed. The best that we reached using the algorithm in this set up was just overfit the model over the source data but not in the target.

B. Dataset SetUp

Another possibility is that given that the classes used in the DeepJDOT paper were only 10 it was easy to be sure that each batch from the source dataset contained all the possible classes. In this setting, since the target classes must be contained in the source classes at each batch all target images could be correctly correlated with one image in the source batch.

This setup was impossible to implement with the amount of classes that our dataset had. With 775 different classes it was too memory expensive to load a batch of that size to be sure that all the classes were represented in the batch. This could have led to a miss mapping of some target examples into bad classes in the source making it difficult for the classifier to learn the correct classes.

C. Hyperparameters Search

The last possibility was the amount of hyperparameters to finetune and the sensibility of the model to each of them. Unfortunately, the paper didn't mention neither the parameters with the ones they runned their experiments nor the range in which the hyperparameters should move. This makes the

hyperparameters search hard and it turns more hard if each training is slow, that is the case when you have several data to process.

In our case we ran different experiments doing a random search and, although we improved the results by fine tuning some of the parameters they were not as good as expected.

On the other hand, the model was really sensible to the hyperparameters. In some experiments we passed from forgetting all the learning and obtaining 0.0 accuracy on the source and the target dataset to obtaining robustness on the source dataset (but no in the target) moving only one hyperparameter.

All these could be possible reasons for the obtained results and to fix them could be some starting points to improve them. The first and second case could be tested implementing the algorithm on a new smaller subset of the dataset. In that case we could select only some of the classes so batches including all of them could be loaded into memory. The last point could be solved with more random search of the hyperparameters that leads to better results.

VII. Conclusions

On this project we presented the results for the implementation of the DeepJDOT for the task of vehicle re-identification using the VeRi Dataset. To do so, we divided the dataset into two sets with different distributions, a source dataset and a target dataset. We trained a convolutional neural network model in a supervised fashion using the source dataset obtaining good results for the source test set. However we showed that the model was not enough to generalize to a dataset with a different distribution (the target) and we proposed that the implementation of the optimal transport for domain adaptation would improve the results. After training the model using the DeepJDOT algorithm, our model was robust achieving almost the same performance over the source dataset but we didn't achieve good enough results in the target. This could be mainly explained because of the hyperparameters of the algorithm or the complexity of the dataset. We analysed each case and proposed some ideas to try in the search for improvements.

References

[1] C. Villani, "Optimal transport, old and new, grundlehren der matematischen wissenschaften, 338 (2008)."

[2] B. B. Damodaran, B. Kellenberger, R. Flamary, D. Tuia, and N. Courty, "Deepjdot: Deep joint distribution optimal transport for unsupervised domain adaptation," in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018, pp. 447–463.

[3] N. Courty, R. Flamary, A. Habrard, and A. Rakotomamonjy, "Joint distribution optimal transportation for domain adaptation," *arXiv preprint arXiv:1705.08848*, 2017.

[4] M. Ye, J. Shen, G. Lin, T. Xiang, L. Shao, and S. C. Hoi, "Deep learning for person re-identification: A survey and outlook," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2021.

[5] H. Wang, J. Hou, and N. Chen, "A survey of vehicle re-identification based on deep learning," *IEEE Access*, vol. 7, pp. 172 443–172 469, 2019.

[6] M. H. Liu X., Liu W., "Large-scale vehicle re-identification in urban surveillance videos," *International Conference on Multimedia and Expo, IEEE*, 2016.

[7] S. R. J. S. Kaiming He, Xiangyu Zhang, "Deep residual learning for image recognition," 2015.

[8] F. Chen, N. Chen, H. Mao, and H. Hu, "Assessing four neural networks on handwritten digit recognition dataset (mnist)," *arXiv preprint arXiv:1811.08278*, 2018.