DOMAIN ADAPTATION FOR DEEP ARCHITECTURES

LE QUOC TUNG
Supervised by Prof. MATTHIEU CORD

December 31, 2020

Abstract

Deep Learning is playing a crucial role in many recent advances in Artificial Intelligence and Machine Learning. The deeper the architecture is, the better our model captures complex features and thus, leads to significant improvement of performance. However, to harness the power of such deep architectures to the fullest, the amount of training data, especially labeled data, has to be enormous as well as universal. To overcome this difficulty, Transfer Learning and one of its branch, Domain Adaptation are proposed with the attempt to reduce the dependence of the deep architectures on the huge availability of unbiased data: the training data could be limited to one specific domain but the model works well for other domains. This report will make an investigation on the most studied approaches of this problem, namely *Maximum Mean Discrepancy* and *Generative Adversarial Network* and their applications to classification problems. Moreover, we proposed an additional mechanism to enhance the effects of existing methods. The experiments on a standard Domain Adaptation dataset, **Office31**, prove the effectiveness of our proposal.

1 An introduction to Transfer Learning and Domain Adaptation

1.1 Motivation

In typical supervised machine learning algorithms, we usually require the training and testing sets to share the same distribution so that the theoretical results, which are based on this assumption, hold. In reality, this requirement is not always necessarily verified. In fact, distribution difference is common and caused by changes in location, background, pose, ... Meanwhile, manually labeling a diverse dataset covering all domains is prohibitive. Consequently, the testing result might be significantly degraded, especially when the difference of distributions between the training and testing sets is huge.

As a result, effective methods exploiting the unlabeled data (referred to as semi-supervised learning) or related models (referred to as Transfer Learning (TL)) appear to be desirable

from practical viewpoints. One of the most active sub-field of TL, Domain Adaptation (DA), is proposed to overcome this difficulty.

In general, DA refers to techniques that leverage labeled data from a relevant *source* domain to obtain a good model for unlabeled data from a *target* domain. Usually, the task is assumed to be identical for both domains (i.e. for classification problem, the sets of categories are the same). In the following parts, the definitions of TL and DA will be formally introduced.

1.2 Transfer Learning and Domain Adaptation

A domain $\mathcal{D}$ is defined by:

1. A dimensional feature space $\mathcal{X} \subset \mathbb{R}^d$ with marginal probability distribution $P(\mathbf{X})$;
2. A task $\mathcal{T}$ characterized by a label space $\mathcal{Y}$ and conditional probability distribution $P(Y|X)$.

Consider two different domains:

1. Source domain: $\mathcal{D}^s = \{\mathcal{X}^s, P(\mathbf{X}^s)\}$ with $\mathcal{T}^s = \{\mathcal{Y}^s, P(\mathbf{Y}^s|\mathbf{X}^s)\}$;
2. Target domain: $\mathcal{D}^t = \{\mathcal{X}^t, P(\mathbf{X}^t)\}$ with $\mathcal{T}^t = \{\mathcal{Y}^t, P(\mathbf{Y}^t|\mathbf{X}^t)\}$.

When $\mathcal{D}^s = \mathcal{D}^t$ and $\mathcal{T}^s = \mathcal{T}^t$, the theoretical requirement of tradition ML is met. Therefore, the model which is trained on the source domain is expected to work well on the target domain.

In contrast, the model trained only on the source domain might work poorly (in case $\mathcal{D}^s \neq \mathcal{D}^t$) or might not be applied directly (in case $\mathcal{T}^s \neq \mathcal{T}^t$) if it is only trained on $\mathcal{D}^s$. However, if $\mathcal{D}^s, \mathcal{T}^s$ are somewhat related to $\mathcal{D}^t, \mathcal{T}^t$, the conditional distribution $P(\mathbf{Y}^t|\mathbf{X}^t)$ could be inferred from $\mathcal{D}^s, \mathcal{T}^s$. This is *Transfer Learning*.

Under the assumption that $\mathcal{T}^s = \mathcal{T}^t$ but $\mathcal{D}^s \neq \mathcal{D}^t$, this is where *Domain Adaptation* is applied. Sometimes, the second condition $P(\mathbf{Y}^s|\mathbf{X}^s) = P(\mathbf{Y}^t|\mathbf{X}^t)$ could be relaxed since it is rather strong and does not always hold in reality. In this report, we require only condition $\mathcal{Y}^s = \mathcal{Y}^t$ to be true.

A final remark is that in DA community, people further distinguish between the *unsupervised* case (training data is **labeled** samples in source domain and **unlabeled** samples in target domain) and *semi-supervised* case (additional labeled samples in target domain are provided). In both scenario, the ultimate goal is to produce a model that performs well on **unlabeled** target domain.

The report will strongly emphasize the mechanism of some DA techniques in a particular problem: image classification. While this family of problems appears relatively simple, its application could extend to other complicated ones like object recognition, semantic segmentation,.... Unless there is explicit clarification, it is assumed that we are working on the *unsupervised* DA for image classification problem.

The arrangement of this report will conclude the introduction: sections 2 and 3 will be devoted to introduce two families of DA techniques, *Maximum Mean Discrepancy* and *Generative Adversarial Network*. Section 4 will propose possible upgrade for existing methods as well as a new mechanism which could be widely applied and burst the performance of DA techniques. Experimental results will be shown in section 5.

2 Maximum Mean Discrepancy in Domain Adaptation

In visual applications, the distribution difference between the training and test sets is called *domain shift*. With the effects of domain shift, classical Machine Learning fails to obtain a reasonable result. One of the most studied approaches for this problem is to mitigate the difference between the distributions of source and target domains. More formally, the models are trained so that not only do they work well on the training set, but they also produce *intermediate features* (i.e, for Neural Network, feature maps) and *conditional probability* (i.e, for Neural Network, it is the last linear layer followed by softmax activation) of both domains so that they share the same statistical characteristics.

In fact, DA has a long history of research. In its early studies, relatively shallow methods are dominant [5] [18] [2]. However, with the appearance of AlexNet [9] and later ResNet [8] as well as their phenomenal performance on classification tasks, deep architectures quickly became popular in computer vision in general and DA in particular. The strength of this approach is its simplicity yet effectiveness that it brings. While the deep architectures permit to learn more meaningful and complex features, people have the freedom to add more constraints in the training processes so that the learning algorithm could find a better solution for the tasks. MMD is a family of techniques which utilize the Maximum Mean Discrepancy to reduce the effects of domain shift. In this section, we will briefly look into some well known techniques of this family.

2.1 Maximum Mean Discrepancy and Domain Adaptation

The concept of Maximum Mean Discrepancy and its applications are inspired from the two-sample problem [7], which addresses the question of whether two independent samples are drawn from the same distribution. It is utilized to measure the distance between the distributions of the source and target domain in many DA problems. This section assumes that readers have fundamental knowledge about Reproducing Kernel Hilbert Space (RKHS). Otherwise, a brief introduction to RKHS is available in Appendix A.

Followings are several important definitions:

Mean Embedding and Maximum Mean Discrepancy

Given a RKHS $\mathcal{H}$ on a set X and the corresponding kernel $K : X \times X \to \mathbb{R}$. There exists an implicit feature map $\phi : X \to \mathcal{H}$ that satisfies $K(x, y) = \langle \phi(x), \phi(y) \rangle$ (by theorem A.3 and equation 29).

Definition 2.1. Mean Embedding of a probability
Let Ω and x be the domain and instantiations of X respectively. Then the mean embedding of a probability distribution P on set X endowed by kernel K is:

$$\mu_X(P) = \mathbb{E}[\phi(X)] = \int_{\Omega} \phi(x) dP(x). \tag{1}$$

Definition 2.2. Maximum Mean Discrepancy (MMD)
Let P, Q be two different probability distributions of X, $\mathcal{F}$ be a class of functions $f : X \to \mathbb{R}$. We define the Maximum Mean Discrepancy (MMD) as:

$$D(\mathcal{F}, P, Q) = \sup_{f \in \mathcal{F}} (\mathbb{E}_{x \sim P}[f(x)] - \mathbb{E}_{x \sim Q}[f(x)]). \tag{2}$$

In the statistics literature, these two quantities could be approximated by the empirical estimations which are obtained by replacing the population expectations with empirical expectations computed on the samples of P and Q,

$$\mu_X(P) = \frac{1}{n} \sum_{i=1}^{n} \phi(x_i); \tag{3}$$

$$D(\mathcal{F}, P, Q) = \sup_{f \in \mathcal{F}} \left(\frac{1}{n} \sum_{i=1}^{n} f(x_i^P) - \frac{1}{m} \sum_{j=1}^{m} f(x_j^Q) \right). \tag{4}$$

MMD offers a good condition to determine whether two probabilities are identical or not. Given $\mathcal{F} = \mathcal{C}(X)$ is the set of continuous bounded functions on X, it could be proved that $P = Q$ if and only if $D(\mathcal{F}, P, Q) = 0$ [6]. Nevertheless, such information is hard to access because MMD requires to solve a functional optimization problem, which is not preferable. On the other hand, the formula of the *mean embedding* is much more comfortable to apply. The following theorem provides a way to work with MMD through the *mean embedding*.

Injectivity of distribution in RKHS

Theorem 2.3. *Let $\mathcal{F}$ be a unit ball in a RKHS $\mathcal{H}$, defined on the compact metric space X with associated with kernel $K(.,.)$, P and Q two distributions on X. Given that RKHS $\mathcal{H}$ is universal (i.e, $K(.,.)$ is continuous, $\mathcal{H}$ is dense in $\mathcal{C}(X)$ with respect to the L_∞), then $D(\mathcal{F}, P, Q) = 0$ if and only if $\mu_X(P) = \mu_X(Q)$.*

In summary, the theorem 2.3 says that the function μ_X is injective under the assumption of the RKHS $\mathcal{H}$ is universal. This property does not hold if $\mathcal{H}$ is not universal in general.

Last but not least, the theorem 2.3 allows the MMD to be approximated as following, which is more applicable,

$$
\begin{aligned}
D(\mathcal{F}, P, Q) &= \left\| \frac{1}{n} \sum_{i=1}^{n} \phi(x_i^P) - \frac{1}{m} \sum_{j=1}^{m} \phi(x_j^Q) \right\|^2 \\
&= \frac{1}{n^2} \sum_{i=1}^{n} \sum_{j=1}^{n} K(x_i^P, x_j^P) - \frac{2}{nm} \sum_{i=1}^{n} \sum_{j=1}^{m} K(x_i^P, x_j^Q) + \frac{1}{m^2} \sum_{i=1}^{m} \sum_{j=1}^{m} K(x_i^Q, x_j^Q) \\
&= D(P, Q).
\end{aligned} \tag{5}
$$

Since this formula of MMD is not dependent on $\mathcal{F}$, $D(\mathcal{F}, P, Q)$ could be simplified to $D(P, Q)$. This formula is simple enough to be applicable in DA. In fact, this method is extensively adopted in DA studies, ranging from traditional shallow to deep methods. In the following sections, we will discuss it more carefully.

2.2 Maximum Mean Discrepancy as a regularisation

In Deep Learning, *fine tuning* is the process of taking a network model that has already been trained for a given task, and make it perform a second similar task. The interesting aspect of such deep architectures like AlexNet or ResNet is that the models resulted from fine-tuning their pre-trained versions (notably trained on ImageNet Large Scale Visual Recognition Challenge [13]) are usually better than any shallow method. By using the checkpoint of the pre-trained models, freezing layers or training with relatively low learning rate, the learning algorithm could take advantage of the information extracted from the previous tasks to reduce the training time and potentially obtain a better result.

However, deep architectures still have millions of parameters. Without sufficient amount of data (millions of them), gradient-based learning algorithms easily overfit. In the context of DA, the problem is even more disastrous because the convenient hypothesis of i.i.d data does not hold. MMD comes in handy at this point since it provides us with an elegant solution for the problem of domain shift.

Formally, given two different domains:

1. Source domain: $\mathcal{D}^s = \{\mathcal{X}, \mathcal{Y}, P(\mathbf{X}, \mathbf{Y})\}$;
2. Target domain: $\mathcal{D}^t = \{\mathcal{X}, \mathcal{Y}, Q(\mathbf{X}, \mathbf{Y})\}$.

It is worth mentioning that this formalization is specific for our problem of interest: unsupervised Domain Adaptation for image classification. To be more specific, while $\mathcal{D}^s$ and $\mathcal{D}^t$ share their data and task spaces ($\mathcal{X}$ and $\mathcal{Y}$), they are characterized by two joint distributions P and Q (so that we no longer have to take care of two distributions $P(\mathbf{X})$ and $P(\mathbf{Y}|\mathbf{X})$ separately). We generally assume that $P \neq Q$.

To tackle this problem, we start from the case where $P = Q$, a conventional machine learning problem. The goal is to learn a function $f : \mathcal{X} \to \mathcal{Y}$ (in this case, a neural network) so that the function $E = \mathbb{E}_{(x,y)\sim P}[\mathbb{1}_{f(x)\neq y}]$ is minimized. Under the assumption that $P = Q$, learning the model from optimizing E on the training set is sufficient to bound the error on the testing case.

To compensate for the missing assumption $P = Q$, many studies have a tendency to rewrite the desirable function as $f(x) = g(h(x))$. In fact, $h : \mathcal{X} \to \mathcal{Z}$ acts as the feature extractor whose job is to learn the rich representation of $\mathcal{X}$. Meanwhile, $g : \mathcal{Z} \to \mathcal{Y}$ is a classifier which plays its role on the $\mathcal{Z}$ space instead of the original $\mathcal{X}$. Since h is not necessarily injective, we could match the distributions $P(h(\mathbf{X}), \mathbf{Y})$ and $Q(h(\mathbf{X}), \mathbf{Y})$. We can apply the theoretical results to g.

However, it is noticeable that the distribution $P(h(\mathbf{X}), \mathbf{Y})$ is too difficult to estimate. Therefore, people sometimes relax the strong assumption $P(h(\mathbf{X}), \mathbf{Y}) = Q(h(\mathbf{X}), \mathbf{Y})$ and aim to achieve a weaker one $P(h(\mathbf{X})) = Q(h(\mathbf{X}))$. In fact, this relaxation was a dominant approach for the MMD-based methods in DA before people came up with a solution to access the joint distribution of $h(X)$ and Y.

Figure 1 shows a typical diagram of a MMD-based neural network model. The network architecture is based on a CNN (like AlexNet or ResNet). On top of their last convolutional layers, there are $n-1$ fully connected layers followed by a linear classifier (one fully connected layer with softmax activation function). With CNNs (like AlexNet, VGG16 [14], VGG19 [14]) whose architectures have several fully connected layers in the end, these layers could be reused to reduce the number of parameters. CNNs which do not have this feature (like ResNet) could be added some layers after their convolutional layers. To facilitate the notation, let $Z_1, Z_2, ..., Z_{n-1}$, which are the outputs of first $n-1$ fully connected layers, be *intermediate features*, Z_n (output of the linear classifier) be *conditional probability*. This notation is natural since $\mathbf{Z} = Z_n$ is usually interpreted as $P(\mathbf{Y}|\mathbf{X})$. Aside from the traditional cross entropy classification loss:

$$\mathcal{L}_c = \frac{1}{N} \sum_{i=1}^{N} \sum_{j=1}^{M} -\mathbb{1}_{y_i=j} \log(Z_{i,j}) \tag{6}$$

where N, M is the number of instances and categories respectively, we also have another type of loss. This loss function is used to enforce the equality $P(h(\mathbf{X}), \mathbf{Y}) = Q(h(\mathbf{X}), \mathbf{Y})$. In other words, the loss function should depend on the samples from the two domains to measure the distance between the two distributions. MMD could be applied on intermediate layers to enable such metric:

$$\begin{aligned} \mathcal{L}_{MMD} &= D(P, Q) \\ &= \frac{1}{n^2} \sum_{i=1}^{n} \sum_{j=1}^{n} K(x_i^s, x_j^s) - \frac{2}{nm} \sum_{i=1}^{n} \sum_{j=1}^{m} K(x_i^s, x_j^t) + \frac{1}{m^2} \sum_{i=1}^{m} \sum_{j=1}^{m} K(x_i^t, x_j^t) \end{aligned} \tag{7}$$

Therefore, the total loss is formulated as (λ is a hyperparameter):

$$\mathcal{L}_{total} = \mathcal{L}_c + \lambda \mathcal{L}_{MMD}. \tag{8}$$

With such a general diagram, it is left to determine the kernel K and the representations $Z_1, Z_2, \ldots, Z_n$ to have a concrete method.

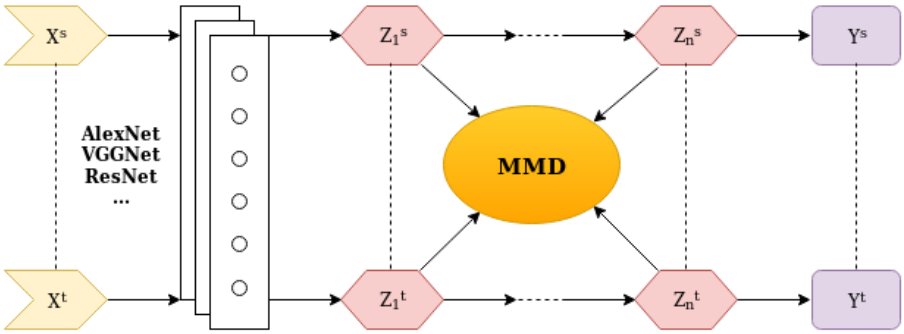

Figure 1: A general architecture of domain adaptation. The input from $\mathcal{X}$ is fed to the original CNN. The intermediate layers like $Z_1, Z_2, \ldots, Z_n$ are not safely transferable. They will be adapted by minimization of the MMD criteria.

2.3 Applications of Maximum Mean Discrepancy

MMD-based loss is featured in many studies of DA. This report will introduce three methods from this family, namely *Deep Domain Confusion* (**DDC**) [17], *Deep Adaptation Network* (**DAN**) [10] and *Joint Adaptation Network* (**JAN**) [12]. Their difference illustrates the impacts of kernel choice and intermediate layers's representation exerting on the model's performance. Figure 2 illustrates the intermediate layers and kernel choice of three methods.

2.3.1 Deep Domain Confusion (DDC)

DDC chooses Linear kernel $K(x_1, x_2) = x_1^T x_2$ for their MMD. The architecture is based on AlexNet, but it has an additional *bottleneck* layer between fully connected layers *fc7* and *fc8* of AlexNet. The loss MMD is thus:

$$\begin{aligned}\mathcal{L}_{MMD} &= \frac{1}{n^2}\sum_{i=1}^{n}\sum_{j=1}^{n}(x_i^s)^T x_j^s - \frac{2}{nm}\sum_{i=1}^{n}\sum_{j=1}^{m}(x_i^s)^T x_j^t + \frac{1}{m^2}\sum_{i=1}^{m}\sum_{j=1}^{m}(x_i^t)^T x_j^t \\ &= \left\| \frac{1}{n}\sum_{i=1}^{n} x_i^s - \frac{1}{m}\sum_{j=1}^{m} x_j^t \right\|_2^2\end{aligned} \tag{9}$$

In short, when learning the CNN on the training data, DDC attempts to minimize the difference between the average representations of the intermediate layers of both domains. The idea is natural yet naive. It is quite effective if there is extremely limited data or in the context of *supervised* domain adaptation.

2.3.2 Deep Adaptation Network (DAN)

Linear combination of Gaussian Kernels $K_{DAN}(x_1, x_2) = \sum_{i=1}^{M} \exp \frac{\|x_1 - x_2\|_2^2}{2\sigma_i^2}$ is the choice of DAN. AlexNet is the basic architecture while the intermediate layers are *fc6*, *fc7* and *fc8*.

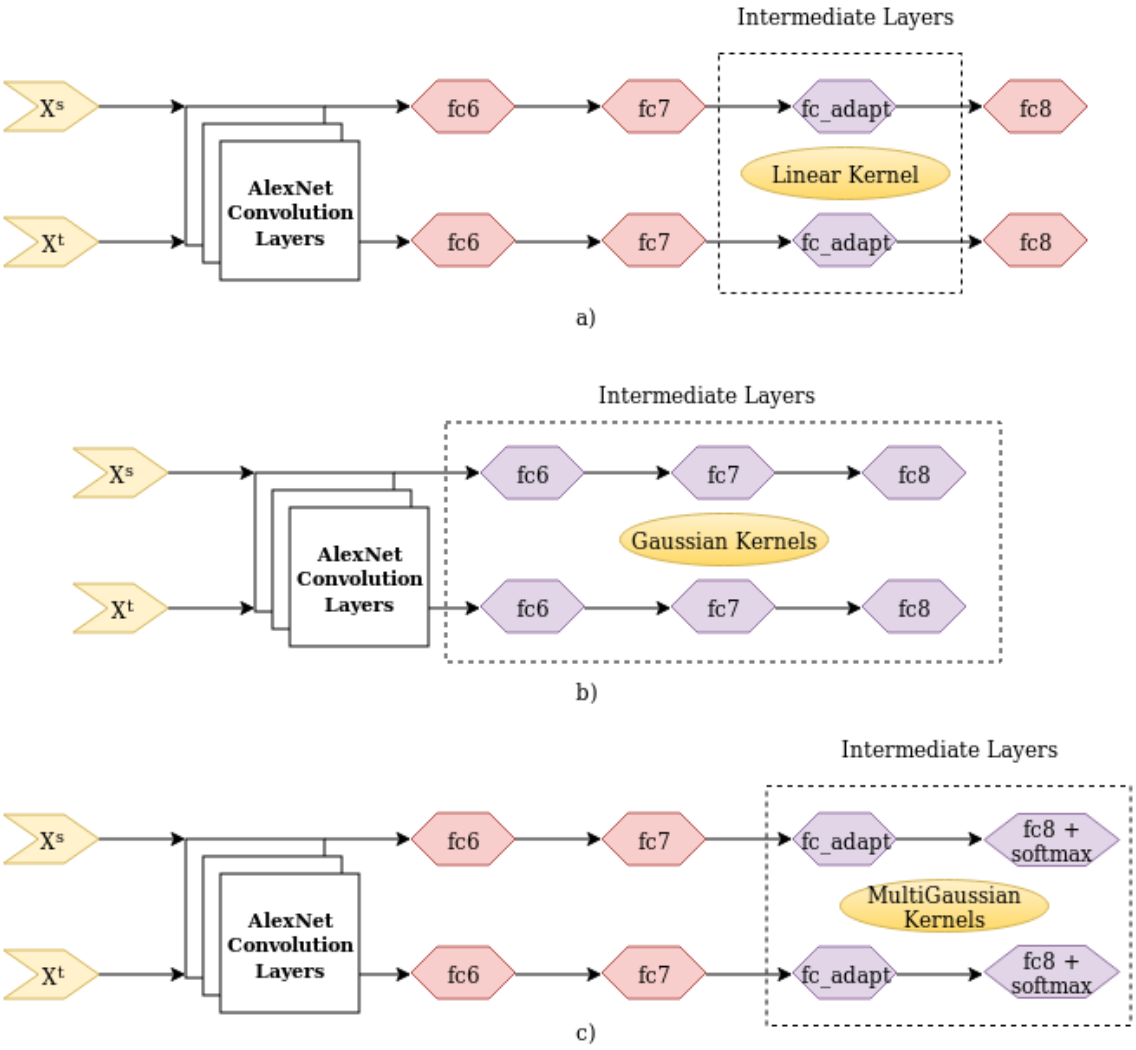

Figure 2: a) DDC on AlexNet with Linear Kernel and a bottleneck layer as intermediate layer b) DAN on AlexNet with Gaussian Kernels and *fc*6, *fc*7, *fc*8 as intermediate layers c) JAN on AlexNet with Gaussian Kernels and tensor product between a bottleneck layer and *fc*8 as intermediate layer and conditional probability layers.

The loss MMD for each layer k is formulated as:

$$\begin{aligned}\mathcal{L}_{MMD}^{l_k} = &\frac{1}{n^2}\sum_{i=1}^{n}\sum_{j=1}^{n} K_{DAN}(x_i^{s,l_k}, x_j^{s,l_k}) - \frac{2}{nm}\sum_{i=1}^{n}\sum_{j=1}^{m} K_{DAN}(x_i^{s,l_k}, x_j^{t,l_k}) \\ &+ \frac{1}{m^2}\sum_{i=1}^{m}\sum_{j=1}^{m} K_{DAN}(x_i^{t,l_k}, x_j^{t,l_k}).\end{aligned} \tag{10}$$

The total MMD loss is:

$$\mathcal{L}_{MMD} = \mathcal{L}_{MMD}^{l_6} + \mathcal{L}_{MMD}^{l_7} + \mathcal{L}_{MMD}^{l_8}. \tag{11}$$

The huge upgrade from DAN to DDC is the adoption of Gaussian kernels. Gaussian kernel has the universality property as required in theorem 2.3, while Linear kernel does not. The adaptation is enhanced by the usage of multiple layers. It is also shown in experimental results that DAN performs better than DDC in the context of *unsupervised* domain adaptation.

2.3.3 Joint Adaptation Network (JAN)

While both previous methods enable the enhancement in the adaptation process, they share the same weakness: the loss MMD is bound to match the distributions $P(h(\mathbf{X}))$ and $Q(h(\mathbf{X}))$ instead of $P(h(\mathbf{X}), \mathbf{Y})$ and $Q(h(\mathbf{X}), \mathbf{Y})$. JAN proposed a representation of the joint distribution $P(h(\mathbf{X}), \mathbf{Y})$. It is worth recalling that the output of the last fully connected layers followed by softmax activation is usually interpreted as the conditional probability $P(\mathbf{Y}|\mathbf{X})$. Therefore, the tensor product of other intermediate layers l_i and the conditional probability l_p enables a natural representation of joint distribution. Moreover, given K_1 and K_2 are the kernel used on l_i and l_p respectively, the natural kernel on the tensor product of l_i and l_p is calculated as:

$$K_{JAN}((x_1, y_1), (x_2, y_2)) = K_1(x_1, x_2)K_2(y_1, y_2). \tag{12}$$

In JAN, K_1 and K_2 are K_{DAN}. Moreover, l_i is a *bottleneck* layer (like DDC) and l_p is the output of last fully connected layer followed by softmax activation. This method appears to be superior than previous ones which could only handle the marginal distributions $P(h(\mathbf{X}))$ and $Q(h(\mathbf{X}))$. Its idea of representation is further harnessed in many following works.

3 Generative Adversarial Network in Domain Adaptation

DA methods in general and the family of MMD-based techniques in particular share the same general strategy: the model needs to compensate for the missing assumption $P = Q$. The approach of MMD is to work on the statistical aspects of source and target data. Despite its nice theoretical property, several drawbacks could be observed:

1. To obtain good approximation of MMD, the source and target data should be large, which is not always the case;

2. MMD computation is quadratic in batch size (assuming we train with SGD-based algorithm). Although [16] and [10] use a linear approximation of equation 10, it might increase the approximation error.

Generative Adversarial Network (GAN) [4] is another approach which facilitates the job of matching two distributions P and Q (which are usually inaccessible) and is easily trained with SGD-based algorithm. In fact, it adopts the concept of *Discriminator* from the original GAN [4], whose job is to distinguish data from two domains. The competition between the original network (image classifier) and *Discriminator* allows the quality enhancement of both networks. In this section, we shortly introduce the GAN, the main scheme of this approach and one of its realization, Conditional Deep Adversarial Network (CDAN) [11].

3.1 Generative Adversarial Network

Generative Adversarial Network or GAN is a framework for estimating the probability distribution over data encountered in artificial intelligence problems. Unlike its counterpart - conditional model, which already helped us to achieve many significant breakthroughs, generative model remained undeveloped until a couple of years ago. The difficulty lies in the intractability of those probabilistic computations that usually involve maximum likelihood estimation. GAN is an elegant solution to overcome this difficulty.

In a typical GAN, there are two neural networks whose names are *Generator* G and *Discriminator* D respectively. While G attempts to learn data distribution p_{data}, D classifies the domains for images generated by G and true data. Both networks are controlled so that they compete with each other and eventually improve their effectiveness in their own tasks. More specifically, let X be the data space and Z be another space. To learn the *Generator*'s distribution $p_g(x)$ over data $x \in X$, we define a prior on input noise variables $p_z(z), z \in Z$ and $G : Z \to X$ is a function mapping to data space. Meanwhile, $D : X \to [0, 1]$ is a classifier which determines whether a sample $x \in X$ comes from the data rather than p_g. Therefore, one could view GAN as a min-max game where two players, G and D, try to minimize/maximize the common objective function $V(G, D)$:

$$V(G, D) = \min_G \max_D \mathbb{E}_{x \sim p_{data}(x)}[\log D(x)] + \mathbb{E}_{z \sim p_z(z)}[\log(1 - D(G(z)))] \tag{13}$$

As a result of the training, we expect the distribution learned by G to be identical to the data distribution. The existence of D, which is unnecessary after training, is to force p_g to continuously improve and eventually become indistinguishable with p_{data}. Indeed, the following theorem confirms this intuition:

The global optimal $p_g = p_{\text{data}}$ [4]

Theorem 3.1. *The global optimal of the virtual training criterion V(G,D) is achieved if and only if:*

1. $D(x) = \frac{p_{data}(x)}{p_{data}(x)+p_g(x)}$;
2. $p_g = p_{data}$.

In that case, $V(G, D) = -\log 4$.

The theorem concludes the brief introduction to GAN. Its application in DA will be introduced in the following sections.

3.2 The application of Generative Adversarial Network in Domain Adaptation

In fact, DA techniques involved with *Discriminator* are quite popular. [16], [3] applied this idea to various DA problems. They work really well and usually outperform MMD-based methods. Its superiority results from the key idea of the original GAN: when we allow the *Generator* and *Discriminator* play the min-max optimization game, they evolve continuously and the *Generator*'s samples become eventually indistinguishable from the data distribution. In the DA scenario, the *Discriminator* allows us to match the distributions of features of source and target domains.

The general architecture of a model incorporated with a GAN is illustrated in Figure 3. Let **G** and **D** be the image classifier and the discriminator respectively. While **G** produces the intermediate layers $\mathbf{Z} = \{Z_1, Z_2, ..., Z_n\}$ and predicts the label **Y**, **D** tries to classify **Z** into 2 domains. Intuitively, the training on **G** and **D** has to achieve the following objectives:

1. **G** works well in (label) classification tasks (on source domain);
2. **G** confuses **D** as much as possible (or **G** simply makes $P(\mathbf{Z}) = Q(\mathbf{Z})$);
3. **D** works well in (domain) classification tasks;

To avoid confusion, let $G_1 : \mathcal{X} \to \mathcal{Z}$ be feature extractor and $G_2 : \mathcal{X} \to \mathbb{R}^M$ (M is the number of categories) be the classifier. G_1 and G_2 are *subfunctions* of **G** (in a sense that they used parameters of **G**). Then the objectives are formally formulated as:

$$\min_{G_2} \mathcal{L}_c = \frac{1}{n_s} \sum_{i=1}^{n_s} \sum_{j=1}^{M} -\mathbb{1}_{y_i=j} \log(G_2(x_i^s)_j); \tag{14}$$

$$\min_{G_1} \max_{D} \mathcal{L}_{ad} = \frac{1}{n_s} \sum_{i=1}^{n_s} \log D(G_1(x_i^s)) + \frac{1}{n_t} \sum_{i=1}^{n_t} \log(1 - D(G_1(x_i^t))). \tag{15}$$

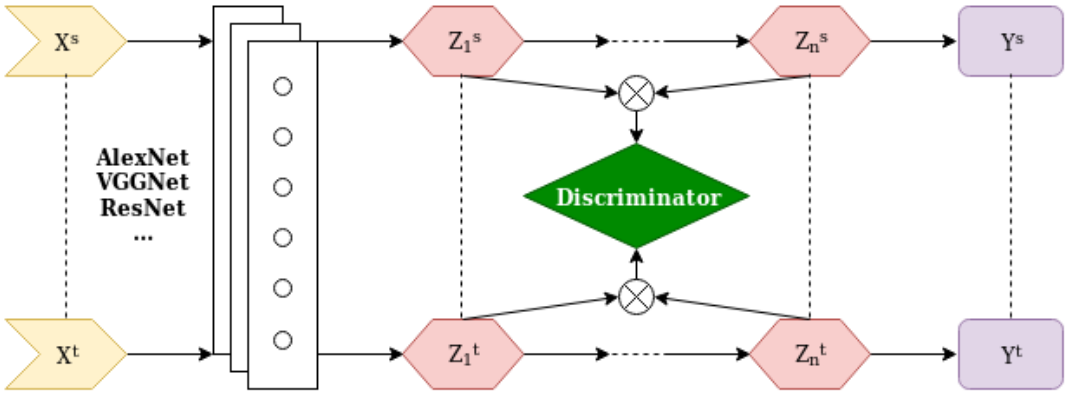

Figure 3: Scheme of Generative Adversarial Network on Domain Adaptation problem

CDAN is a realization of this general scheme. In fact, CDAN reuses the idea of JAN - tensor product between intermediate layers and conditional probability layers as data representation. This representation is used as input to **D**, which is a multi-layer neural network.

4 Improvement for existing methods

In this section, three main contributions of this report are introduced:

1. **Discussion of Kernel Choice**: This proposal is for MMD-based method. We discuss its potential and propose two different kernels which could outperform the existing ones.
2. **Choice of Input Representation for GAN**: GAN-based approaches benefit from this proposal. Our attempt is to replace the tensor product between layers by a bilinear BLOCK model. Since the rich representation of bilinear model is used as the input of the *Discriminator*, it would help to improve the performance of the *Discriminator* as well as the classifier.
3. **Proposal of Block Adaptation**: A family of mechanisms whose main concept is to learn two distinct classifiers for source and target domains. It leads to significant enhancement in model performance as shown in the experimental result.

4.1 A different kernel choice

Following the flow of ideas from DDC to JAN, we could see two main important improvements:

1. Kernel choice: From Linear to Gaussian kernel, a universal one which has a nice theoretical result in 2.3;

2. Distribution representation: From marginal distribution $P(h(\mathbf{X}))$ and $Q(h(\mathbf{X}))$ to joint distribution $P(h(\mathbf{X}), \mathbf{Y})$ and $Q(h(\mathbf{X}), \mathbf{Y})$.

Consequently, we have two corresponding expansions which could further enhance the performance on DA problem:

1. A better kernel choice: Is Gaussian kernel the best choice? Or is the choice of kernel dependent on the dataset and original architecture (AlexNet, VGG or ResNet)? In fact, Gaussian kernel appears generic and could be replaced with a better one.

2. A better distribution representation: tensor product is a natural representation. Is there a richer representation for $(h(\mathbf{X}), \mathbf{Y})$?

The latter will be discovered in the following section. Here, we focus on the first potential improvement: kernel choice. It is worth re-emphasizing that the Gaussian kernel has the universal property, which allows the mean embedding to be injective. We also want to propose new kernels with the same property.

Secondly, observation from Gaussian kernel shows that the norm $L2$ in the exponent seems not to be adaptive with the conditional probability layers, where its elements are positive and smaller than 1 (since their sum is equal to 1). A norm like $L1$ could be more effective in our speculation. In fact, consider x_1, x_2 to be two vectors of conditional probability layers, $\|x_1 - x_2\|_1$ is the total variation distance of probability.

Combining those two observations, we propose to adopt the Laplace kernel, whose formula is:

$$K_{Laplace}(x_1, x_2) = \exp \frac{\|x_1 - x_2\|_1}{\gamma} \tag{16}$$

Not only is the Laplace kernel universal but it also adopts our proposal of norm $L1$. Although Laplace kernel is only used on the conditional probability layer (last fully connected layers with softmax activation) at first, this kernel is also working well with intermediate feature layers. In the experiments, we used the paradigm of JAN, with two different newly-crafted kernels, namely *Joint Gaussian - Laplace Adaptation Network* (JGLAN) and *Joint Laplace - Laplace Adaptation Network* (JLLAN) whose formulas are:

$$K_{JGLAN}((x_1, y_1), (x_2, y_2)) = \sum_{i=1}^{N} \exp \frac{\|x_1 - x_2\|_2^2}{2\sigma_i^2} \sum_{j=1}^{M} \exp \frac{\|y_1 - y_2\|_1}{\gamma_j} \tag{17}$$

$$K_{JLLAN}((x_1, y_1), (x_2, y_2)) = \sum_{i=1}^{N} \exp \frac{\|x_1 - x_2\|_1}{\gamma_{1,i}} \sum_{j=1}^{M} \exp \frac{\|y_1 - y_2\|_1}{\gamma_{2,j}} \tag{18}$$

where N, M are the number of Gaussian (or Laplace) kernels in the linear combination. The effects of these kernels will be extensively studied in the experimental section.

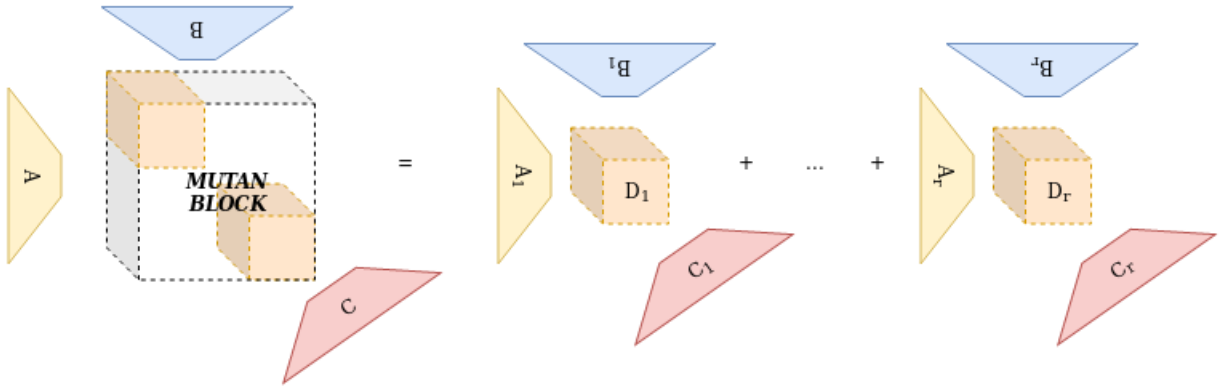

Figure 4: BLOCK model - block decomposition

.

4.2 Deep Adversarial Network with BLOCK fusion model

In this part, we introduce the concept of BLOCK [1] and our BLOCK fusion strategy. BLOCK is one of the most successful approaches for solving two challenging problems: Visual Questioning Answering (VQA) and Visual Relationship Detection (VRD). In those problems, decision making strategies have to take into consideration two or more modalities. The BLOCK model provides us a fusion mechanism which could construct rich and complex representation.

This strength of BLOCK model appears appealing in our problem, especially GAN-based methods. Recalling in CDAN, the input of the *Discriminator* is the tensor product between intermediate layers and conditional probability layers. Although this representation is simple and fairly effective, a richer one might greatly benefit the model. BLOCK model could be taken advantage to fully harness the relation between layers, thus resulting into profound representation and higher accuracy for classifier.

In fact, BLOCK is a bilinear model, which takes as input two vectors $x^1 \in \mathbb{R}^I$ and $x^2 \in \mathbb{R}^J$ and projects them into a K-dimensional space.

$$y = \mathcal{T} \times_1 x^1 \times_2 x^2 \tag{19}$$

where $y \in \mathbb{R}^K, \mathcal{T} \in \mathbb{R}^{I \times J \times K}$.

Therefore, for each $k \in [1, K]$, each term $y_k = \sum_{i=1}^{I} \sum_{j=1}^{J} \mathcal{T}_{ijk} x_i^1 x_j^2$, is a quadratic form of the input.

However, directly applying the idea of BLOCK causes huge problem when the model is trained: the total number of parameters for BLOCK is $I \times J \times K$, which could be huge. This greatly slows down the training process as well as causes potential overfitting. In this report, an alternative BLOCK fusion model is adopted from [1] and illustrated by Figure 4. In fact, $\mathcal{T}$ is expressed using block-term decomposition. Such decomposition of rank (L, M, N) is formulated as:

$$\mathcal{T} = \sum_{i=1}^{r} \mathcal{D}_i \times_1 \mathcal{A}_i \times_2 \mathcal{B}_i \times_3 \mathcal{C}_i \tag{20}$$

where $\mathcal{D}_i \in \mathbb{R}^{L \times M \times N}, \mathcal{A}_i \in \mathbb{R}^{I \times L}, \mathcal{B}_i \in \mathbb{R}^{J \times M}, \mathcal{C}_i \in \mathbb{R}^{K \times N}$.

The total parameter is $r(LMN + IL + JM + KN) \ll IJK$ since $r, L, M, N \ll I, J, K$ in reality. Moreover, this decomposition poses a rank constraint for bilinear formula, which could also control the model complexity.

In our model, x_1 and x_2 are the *intermediate layers* and *conditional probability* respectively. The result of BLOCK will be used as the input for the *Discriminator* in CDAN instead of their simple tensor product.

4.3 Block Adaptation

Though two different approaches, MMD and GAN, which are introduced in previous sections, are distinct in the way of compensation for the assumption i.i.d of data distribution, their ultimate goal is to learn a unique function $F : \mathcal{X} \to \mathcal{Y}$ which works well for both domains. While the idea is simple and intuitive, forcing a neural network to learn one function accomplishing multiple objectives at the same time might not give the optimal result. In this part, an alternative approach is proposed to address this issue.

Our proposal named *Block Adaptation*, whose main scheme is illustrated in Figure 5. To be more specific, after convolutional layers and before the final linear classifier layer, the architecture is inserted with three parts, whose names are Z_{source}, Z_{target} and Z_{common} respectively. Those three components together construct a *Block Adaptation*. While the data from both domains shares the same parameters in Z_{common} and outside *Block adaptation*, each of them uses parameters in Z_{source} or Z_{target} on their own to compute their corresponding results of the block.

In fact, the main difference in our proposal comparing to the traditional one is the learning of two different functions F_s, F_t instead of only one function F. The difference between Z_{source} and Z_{target} allows this mechanism. Then, the learned F_s and F_t are good classifiers for source and target domain respectively. Intuitively, F_s and F_t should satisfy the following statements to result in a good performance:

1. F_s must be a good classifier for source domain.
2. F_s and F_t are *close* since they share the tasks. The definition of *closeness* between F_s and F_t will be extensively elaborated when we have a concrete architecture.
3. F_s and F_t produce intermediate layers sharing the same statistical aspect (i.e $P(h(\mathbf{X}), \mathbf{Y}) = Q(h(\mathbf{X}), \mathbf{Y})$).

While the first condition is guaranteed by the training process on labeled source data, the third condition could be satisfied by the adoption of either GAN or MMD. That leaves the second condition to be under investigation. In the report, we propose 2 realizations of this general *Block Adaptation* scheme, each of which possess different methods to ensure the *closeness* condition.

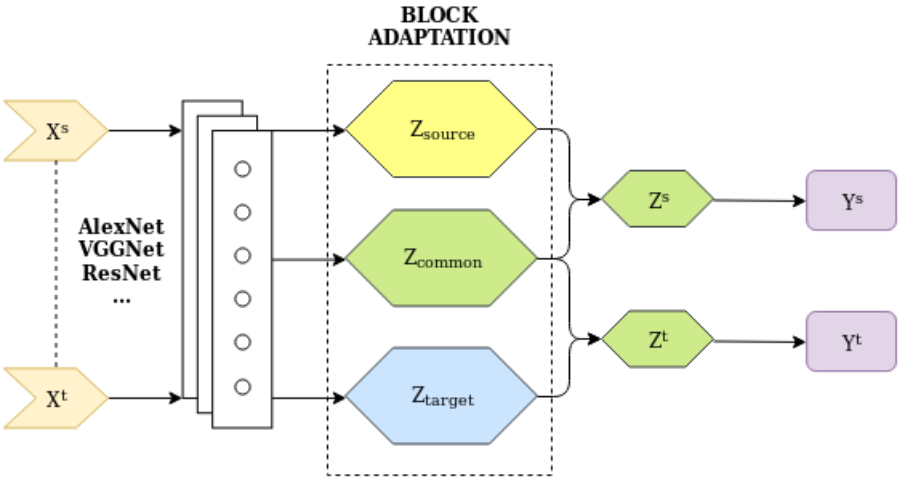

Figure 5: General scheme for a Block Adaptation

4.3.1 Residual Block with different contribution on skip connection path

Residual block [15] in ResNet is the inspiration led to the realization of *Block adaptation*. In a typical residual block, there are two paths (or two functions):

1. Main path (F_1): contain (multiple) fully connected (or convolutional) layers.
2. Skip connection path (F_2): contain an identity map (if the input and output share the same dimension) or a projection map (linear transformation) (if the input and output have different dimension). In this report, we only consider the latter.

The final result of a residual block is $F(x) = F_1(x) + F_2(x)$. Residual network and its component, residual block, succeed possibly thanks to the reformulation of the optimization problem in the training, which results to a more compatible one with SGD-based algorithm. Nevertheless, our *Block adaptation* does not share this intuition with residual block.

In fact, the *Block adaptation* utilizes parameters in $F_1(x)$ as Z_{common}, the shared component between the two domains. The component $F_2(x)$, however, will cause the difference in the processing of two domains since it uses Z_{source} for source domain and Z_{target} for target domain. Thus, after passing *Block adaptation*, data from two domains are treated differently, which leads to the establishment of two distinct functions F_s and F_t. In the first strategy, we apply a scalar multiplication before adding $F_2(x)$ in target domain. Hence, the formula to calculate the intermediate layers for source (Z^s) and target (Z^t) domain are:

$$F_s(x) = F_1(x) + \alpha F_2(x); \quad (21)$$
$$F_t(x) = F_1(x) + F_2(x). \quad (22)$$

where α is a hyper-parameter. That implies $Z_{target} = \emptyset, Z_{source} = \alpha(\alpha \neq 1)$ and other parameters are shared.

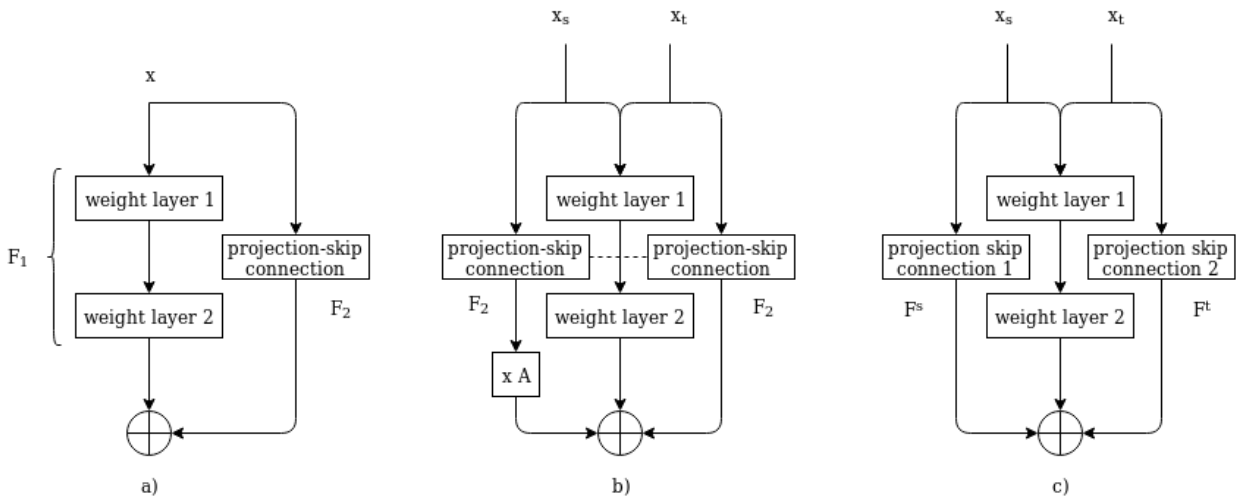

Figure 6: a) Traditional Residual Block. b) Residual Block with different contribution on skip connection path. c) Residual Block with different projection on skip connection path

4.3.2 Residual Block with different projection on skip connection path

Another configuration for *Block adaptation* is to change the projection matrix in the skip connection path. The corresponding equations for $F_s(x)$ and $F_t(x)$ are:

$$F_s(x) = F_1(x) + W_s x + b_s; \tag{23}$$
$$F_t(x) = F_1(x) + W_t x + b_t. \tag{24}$$

In this case, $Z_{source} = \{W_s, b_s\}, Z_{target} = \{W_t, b_t\}$. However, unlike previous configuration, there are many parameters which are different. The closeness between the two functions F_s and F_t is not guaranteed. Thus, it needs to be explicitly enforced by the loss function. We proposed a simple form for the closeness between F_s and F_t:

$$\mathcal{L}_{closeness} = \|W_s - W_t\|_2^2 + \|b_s - b_t\|_2^2. \tag{25}$$

Therefore, the final loss function has the following formula:

$$\mathcal{L} = \mathcal{L}_c + \alpha \mathcal{L}_{MMD/GAN} + \mathcal{L}_{closeness}. \tag{26}$$

5 Experiments

5.1 Setup

The dataset which is used in all the experiments is named **Office31**. It comprises 4652 images from three sources (domains), namely **Amazon (A)** (2817 images), **Webcam (W)**

Method	A-W	A-D	W-A	W-D	D-A	D-W	average
DAN + AlexNet	63.90	64.19	49.02	99.13	50.68	95.30	70.37
JAN + AlexNet	70.81	70.74	52.54	98.46	51.03	**96.10**	73.28
JGLAN + AlexNet	71.70	70.95	**53.80**	98.20	**53.29**	95.30	73.88
JLLAN + AlexNet	**72.45**	**71.08**	53.60	**99.26**	52.56	95.89	**74.14**

Table 1: Results and comparison between MMD-based methods

(795 images) and **DSLR (D)** (498 images). We evaluate 6 tasks of domain adaptation: **A** $\rightarrow$ **D**, **A** $\rightarrow$ **W**, **W** $\rightarrow$ **D**, **W** $\rightarrow$ **A**, **D** $\rightarrow$ **A** and **D** $\rightarrow$ **W**.

The experimental protocol is the standard one for **unsupervised** DA: all source data is labeled while target data's label is omitted. The goal is to establish a reasonable model to predict the label for target data.

Aside from existing method (**DAN**, **JAN**, **CDAN**), the results of our proposed methods are also presented. Based on two kernels in equations 17 and 18, there are two MMD-based methods, which are **JGLAN** and **JLLAN** (section 4.1) respectively. The incorporation of BLOCK model to **CDAN** creates a method named **MCDAN** (section 4.2). Finally, with the first (and second) realization of *Block adaptation* adopted into **JAN**, **CDAN**, we have new techniques $\mathbf{BJAN}_1$, $\mathbf{BCDAN}_1$ (and $\mathbf{BJAN}_2$, $\mathbf{BCDAN}_2$) (section 4.3).

All the tasks and methods are implemented (or re-implemented) in the *Tensorflow* framework, version 1.9.0. The same setting guarantees the fairness of the comparison. All the previous DA methods (**DAN**, **JAN**, **CDAN**) are reworked in Tensorflow (their original version is implemented on **Caffe** (if the original network is *AlexNet*) or **Pytorch** (if the original network is *ResNet50*)). Except the change of framework (and consequently the checkpoint of original CNNs), all hyper-parameters remains loyal to the choice of their original works.

To be more specific, the model is fine tuned from ImageNet pre-trained model [13]. For both *AlexNet* and *ResNet*, all convolutional layers are fine-tuned and linear layers are trained from scratch. Learning rate for the newly-trained layers are 10 times higher than those initialized by pre-trained model. Methods which utilize **CDAN** usually have lower learning rate comparing to **MMD** because of the notorious numerical instability of GAN.

α is chosen dependant on the architecture. Generally, with methods based on *AlexNet*, α is changed gradually from 0 to 1 by the formula: $\alpha_p = \frac{2}{1+\exp(-10p)} - 1$, where $p = \frac{\text{current epoch}}{\text{total training epochs}}$. For **CDAN** and its related methods on *ResNet*, it is important to keep α relatively lower, from 0.1 to 0.5 so that the GAN is stable.

In **JGLAN** and **JLLAN**, the implementation adopts the scheme of **JAN**. However, the numbers of kernels in each linear combination of equations 17 and 18 are 5 and 5, instead of 5 and 1. γ and σ are the average pairwise distance on the training data - the *median heuristic* [7].

5.2 Results and Analysis

Firstly, our attention is on MMD-based methods. From the observation of Table 1, it is clear that our newly-proposed kernels (**JGLAN**, **JLLAN**) outperformed existing ones (**JAN**, **DAN**). It might imply that our intuition of using norm $L1$, which is more sensitive with the

Method	A-W	A-D	W-A	W-D	D-A	D-W	average
CDAN + AlexNet	71.99	69.81	56.08	99.33	55.48	**96.73**	74.90
MCDAN + AlexNet	**73.80**	**70.74**	**56.98**	**99.73**	**56.15**	96.39	**75.63**
CDAN + ResNet	**84.95**	**82.26**	**68.78**	**99.80**	**71.18**	**97.74**	**84.12**
MCDAN +ResNet	84.15	80.99	64.63	99.73	64.21	97.36	81.84

Table 2: Results and comparison between CDAN and MCDAN

Method	A-W	A-D	W-A	W-D	D-A	D-W	average
JAN + AlexNet	70.81	70.74	52.54	98.46	51.03	**96.10**	73.28
$BJAN_1$ + AlexNet	71.86	70.95	**52.57**	99.13	**52.25**	95.72	**73.75**
$BJAN_2$ + AlexNet	**72.83**	**74.76**	50.64	**99.53**	48.76	95.81	73.72
CDAN + AlexNet	71.99	69.81	56.08	99.33	55.48	96.73	74.90
$BCDAN_1$ + AlexNet	73.33	70.81	**56.24**	99.39	**59.06**	**96.93**	**75.97**
$BCDAN_2$ + AlexNet	**74.92**	**72.55**	54.94	**99.93**	55.15	96.31	75.63
CDAN + ResNet	84.95	82.26	68.78	99.80	71.18	97.74	84.12
$BCDAN_1$ + ResNet	85.58	83.80	**69.97**	99.93	**71.57**	97.86	**84.79**
$BCDAN_2$ + ResNet	**86.67**	**85.58**	68.73	**100.0**	67.74	**98.07**	84.51

Table 3: Results and comparison between methods before and after Block Adaptation

shift of conditional distribution, is reasonable. It is also worth noticing that in conditional probability layer, our approach uses the median heuristic γ instead of fixed one (1.68 in **JAN**). The effect hints that kernel should be more adaptive to data. The combination of two changes result in better MMD-based methods.

MCDAN is an interesting case to analyse because it demonstrates the relationship of original architecture and DA technique. From the result shown in Table 2, **MCDAN** on *AlexNet* depicts the superiority of **MCDAN** over **CDAN**. However, an opposite phenomenon is observed when *ResNet* is the basic architecture. Our explanation for this situation is the instability of **GAN**. *ResNet*, which is a more complex network comparing to *AlexNet*, produces features changing rapidly during training process. This makes the *Discriminator*, whose BLOCK model is also more complicated than original **CDAN**, unable to handle properly its task and drives the whole system to collapse. To be able to train **MCDAN** (on both *AlexNet* and *ResNet*), learning rate as well as the trade-off between equations 14 and 15 (section 3.2) have to be low. While this solution works well for *AlexNet*, it is not the case for *ResNet*.

Finally, the analysis on *Block adaptation* will conclude the section. Over all three based DA cases (**JAN**, **CDAN** on *AlexNet*, **CDAN** on *ResNet*) shown by Table 3, our proposed $\mathbf{BJAN}_1$ and $\mathbf{BCDAN}_1$ outperform their corresponding based methods. Meanwhile, $\mathbf{BJAN}_2$ and $\mathbf{BCDAN}_2$ appear better than $\mathbf{BJAN}_1$ and $\mathbf{BCDAN}_1$ respectively in three tasks ($\mathbf{A} \rightarrow \mathbf{W}$, $\mathbf{A} \rightarrow \mathbf{D}$ and $\mathbf{W} \rightarrow \mathbf{D}$). The common thing in these tasks is that the number of images in source domain is higher than that of target domain. However, whether this observation is important or not is still under investigation. Still, the learning of two "close" classifier, each suitable for one domain allows a superior mechanism, as shown in the experimental result.

6 Acknowledgement

I would like to gratefully thank my supervisor Prof. Matthieu Cord for giving me interesting research problems during my internship. The discussion with him opened up many avenues that I would not have been aware of. My thanks are also dedicated to PostDoc. Taylor Mordan and PhD. Antoine Saporta in MLIA LIP6 team for their guidance and comments in the practical experiments as well as the report. Finally, I am deeply grateful to the whole MLIA team, LIP6, especially other PhDs supervised by Prof. Matthieu for their help and gentleness during the internship.

References

[1] Hedi Ben-Younes, Remi Cadene, Nicolas Thome, and Matthieu Cord. Block: Bilinear superdiagonal fusion for visual question answering and visual relationship detection. In *Proceedings of the 33st Conference on Artificial Intelligence (AAAI)*, 2017.

[2] Hal Daumé, III, Abhishek Kumar, and Avishek Saha. Frustratingly easy semi-supervised domain adaptation. In *Proceedings of the 2010 Workshop on Domain Adaptation for Natural Language Processing*, DANLP 2010, pages 53–59, Stroudsburg, PA, USA, 2010. Association for Computational Linguistics.

[3] Yaroslav Ganin, Evgeniya Ustinova, Hana Ajakan, Pascal Germain, Hugo Larochelle, François Laviolette, Mario Marchand, and Victor Lempitsky. Domain-adversarial training of neural networks. *J. Mach. Learn. Res.*, 17(1):2096–2030, January 2016.

[4] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'14, pages 2672–2680, Cambridge, MA, USA, 2014. MIT Press.

[5] Kristen Grauman. Geodesic flow kernel for unsupervised domain adaptation. In *Proceedings of the 2012 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, CVPR '12, pages 2066–2073, Washington, DC, USA, 2012. IEEE Computer Society.

[6] Arthur Gretton, Karsten M. Borgwardt, Malte J. Rasch, Bernhard Schölkopf, and Alexander Smola. A kernel two-sample test. *J. Mach. Learn. Res.*, 13:723–773, March 2012.

[7] Arthur Gretton, Dino Sejdinovic, Heiko Strathmann, Sivaraman Balakrishnan, Massimiliano Pontil, Kenji Fukumizu, and Bharath K. Sriperumbudur. Optimal kernel choice for large-scale two-sample tests. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1205–1213. Curran Associates, Inc., 2012.

[8] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015.

[9] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'12, pages 1097–1105, USA, 2012. Curran Associates Inc.

[10] M. Long, Y. Cao, Z. Cao, J. Wang, and M. I. Jordan. Transferable representation learning with deep adaptation networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pages 1–1, 2018.

[11] Mingsheng Long, Zhangjie Cao, Jianmin Wang, and Michael I Jordan. Conditional adversarial domain adaptation. In *Advances in Neural Information Processing Systems*, pages 1645–1655, 2018.

[12] Mingsheng Long, Jianmin Wang, and Michael I. Jordan. Deep transfer learning with joint adaptation networks. *CoRR*, abs/1605.06636, 2016.

[13] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. Imagenet large scale visual recognition challenge. *Int. J. Comput. Vision*, 115(3):211–252, December 2015.

[14] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. In *International Conference on Learning Representations*, 2015.

[15] Baochen Sun and Kate Saenko. Deep CORAL: correlation alignment for deep domain adaptation. *CoRR*, abs/1607.01719, 2016.

[16] Eric Tzeng, Judy Hoffman, Trevor Darrell, and Kate Saenko. Simultaneous deep transfer across domains and tasks. *CoRR*, abs/1510.02192, 2015.

[17] Eric Tzeng, Judy Hoffman, Ning Zhang, Kate Saenko, and Trevor Darrell. Deep Domain Confusion: Maximizing for Domain Invariance. *arXiv e-prints*, page arXiv:1412.3474, Dec 2014.

[18] Jindong Wang, Wenjie Feng, Yiqiang Chen, Han Yu, Meiyu Huang, and Philip S. Yu. Visual domain adaptation with manifold embedded distribution alignment. In *Proceedings of the 26th ACM International Conference on Multimedia*, MM '18, pages 402–410, New York, NY, USA, 2018. ACM.

A Reproducing Kernel Hilbert Space and Kernel

Reproducing Kernel Hilbert Space (RKHS)

Definition A.1. Let X be an arbitrary set and $\mathcal{H}$ a Hilbert space of real-valued functions on X. The evaluation functional over the Hilbert space of functions $\mathcal{H}$ is a linear functional that evaluates each function at a point x,

$$L_x : f \to f(x), \forall f \in \mathcal{H} \tag{27}$$

Then $\mathcal{H}$ is a **Reproducing Kernel Hilbert Space (RKHS)**, if for all x in X, L_x is continuous at any f in $\mathcal{H}$

It could be proved that $\forall x \in X, \forall f \in \mathcal{H}$, there exists a unique element $K_x \in \mathcal{H}$ such that:

$$f(x) = L_x(f) = \langle f, K_x \rangle \tag{28}$$

Reproducing Kernel

Definition A.2. Let X be an arbitrary set and $\mathcal{H}$ a Hilbert space of real-valued functions on X. For $\forall x \in X$, K_x is identically defined in equation (28), the **Reproducing Kernel (or Kernel)** of $\mathcal{H}$ is a function $K : X \times X \to \mathbb{R}$ formulated by:

$$K(x, y) = \langle K_x, K_y \rangle \tag{29}$$

It could be easily deduced that the kernel K of $\mathcal{H}$ has several following properties:

1. K is symmetric
2. K is positive definite, i.e, $\forall n \in \mathbb{N}, x_1, \ldots, x_n \in X, c_1, \ldots, c_n \in \mathbb{R}$, we have:

$$\sum_{i=1}^{n} \sum_{j=1}^{n} c_i c_j K(x_i, x_j) \geq 0 \tag{30}$$

With those definitions, we can see how to construct a kernel function which is symmetric and positive definite from a RKHS. The Moore-Aronszajn theorem goes the opposite direction. It is stated as follow:

Moore-Aronszajn theorem

Theorem A.3. *For any set X and a function $K : X \times X \to \mathbb{R}$ symmetric and positive definite. There is a unique Hilbert space of functions on X for which K is a kernel.*

Another concept which is important in this section is feature map. Following is its definition

Feature map

Definition A.4. Given a set X and a Hilbert space $\mathcal{H}$ (which we call feature space), a **feature map** is a map $\phi : X \to \mathcal{H}$

A feature map ϕ naturally defines a kernel via: $K(x, y) = \langle \phi(x), \phi(y) \rangle$. Conversely, every positive definite function K and corresponding RKHS $\mathcal{H}$ (the existence of this RKHS is guaranted by A.3) has (infinitely) many associated feature maps, for example, $\phi(x) = K_x$.