Computer Vision

Final Report

Yudhik Agrawal	20161093
Samyak Jain	20161083
Anvesh Chaturvedi	20161094

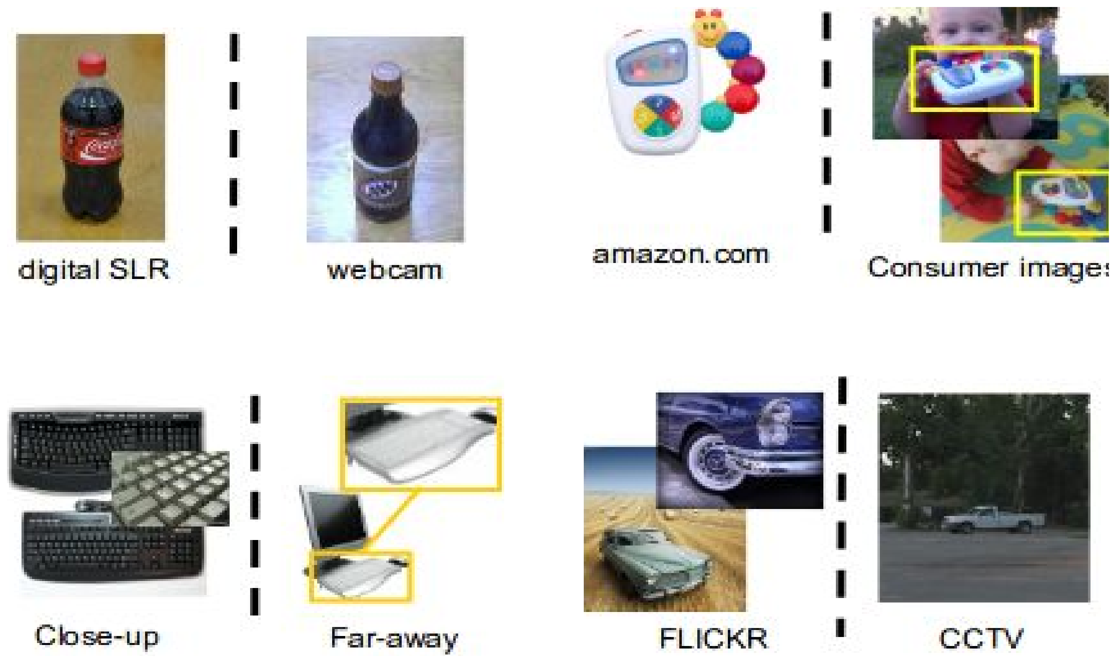
Problem : **Domain Adaptation**

Introduction

Standard supervised learning considers being given data 'x' and labels 'y' drawn from some distribution, 'D' at training time and fits model parameters so as to minimize some loss between prediction labels, 'p' , and the true known labels, 'y'. A crucial assumption in the supervised learning setup is that new test time data, x_{te} , will be drawn from the same distribution ,'D', that was seen at training time. Most guarantees about the performance of a model trained in a supervised way are predicted on this assumption.

Domain adaptation tries to avoid this assumption by operating under the explicit assumption of distribution shift between the training and test domain. In particular there is assumed to be a large labeled source domain dataset,{x,y}, drawn from the distribution X. However, at test time we assume we will receive data from a distinct target domain with data points, v, drawn from a target distribution,V.

Variance in Data in different domains



Goal Of Domain Adaptation

The goal of domain adaptation is to learn to adapt the source model for improved performance in the target domain.

Efficient Learning of Domain-invariant Image Representations 2013

Our project deals with **Efficient Learning of Domain-invariant Image Representations.** The *algorithm* proposed *learns representations* which explicitly ***compensates for domain mismatch*** and which can be efficiently realized as linear classifiers. The ideal image representations does not only depend on the task but also on the domain. It has been *observed* that a *significant degradation in the performance* of *state-of-the-art image classifiers* when ***input feature distributions change*** due to *different image sensors* and *noise conditions*, *pose changes*, *a shift from commercial to consumer video*, and, more generally, ***training datasets biased*** by the way in which they were collected.

Goal

The **goal** of *the project* if to finally achieve **Category Invariant Feature Transform** so that *final classification errors* can be ***minimized***. Multiple approaches can be tried to achieve this. We *present* a ***cohesive framework*** *for learning a single transformation matrix* **W** which *maps examples* between the source and target domains. The *objective* for the *transformation* is to ***diminish domain-induced*** differences so that examples can be compared directly.

Related Works

Approach 1 : Category Invariant Feature Transformations through Similarity Constraints

Learning a transformation can be viewed as ***learning a similarity function*** between *source* and *target points*,.

$$sim(W, x, v) \;=\; x^T W v$$

Intuitively, *a desirable property* of this ***similarity function*** is that it should have a high value when the *source and target points are of the same category* and a *low value* when the ***source and target points*** are of different categories. This approach has been used by some of previous works and works decently.

Approach 2 : Category Invariant Feature Transformations through Optimizing Classification Objective

The ***goal*** in this case is to ***directly optimize a classification objective*** for the target points, while simultaneously presenting *a learning algorithm* that is *more scalable* with *the number of labeled source and target points.* Intuitively, we seek to *learn a transformation matrix* **W** such that once **W** is *applied to the target points*, they will be *classified accurately by the source **SVM***.

Our Approach : Jointly Optimizing Classifier and Transformation

Let $\boldsymbol{x_{s1}}, \boldsymbol{x_{s2}}, .. , \boldsymbol{x_{sn}}$ denote the training points in the *source domain* (**DS**), with labels $\boldsymbol{y_{s1}}, \boldsymbol{y_{s2}}, .. , \boldsymbol{y_{sn}}$. Let $\mathbf{x_{t1}}, \mathbf{x_{t2}}, .. , \mathbf{x_{tn}}$ ***T*** denote the labeled points in the *target domain* (**DT**), with labels $\boldsymbol{y_{t1}}, \boldsymbol{y_{t2}}, .. , \boldsymbol{y_{tn}}$ **T**.
The *approach* presented by the project we are working on has the goal to jointly learn :

1. ***Affine hyperplanes*** that *separate the categories* in the *common domain* consisting of the *source domain* and *target points* projected to the *source*.
2. The *new feature representation* of the *target domain* determined by the ***transformation matrix*** **W** *mapping points* from the *target domain into the source domain*.

We formulate a *joint learning problem* for the ***transformation matrix*** and ***the classifier parameters;*** i.e., ***the hyperplane parameters*** and thus the *decision boundary* are also *affected by the additional training data* provided from the target domain.

The **transformation matrix** should have the property that it ***projects*** *the* ***target points*** *on to the correct side of each* **source**

hyperplane and the **joint optimization** also ***maximizes the margin between two classes***.

For *simplicity of presentation*, the optimization problem for a binary problem with *no slack variables* is as follows :-

$$\begin{aligned} \min_{W,\theta,b} \quad & \frac{1}{2}||W||_F^2 + \frac{1}{2}||\theta||_2^2 \\ \text{s.t.} \quad & y_i^s \left(\begin{bmatrix} x_i^s \\ 1 \end{bmatrix}^T \begin{bmatrix} \theta \\ b \end{bmatrix} \right) \geq 1 \qquad \forall i \in \mathcal{D}_S \\ & y_i^t \left(\begin{bmatrix} x_i^t \\ 1 \end{bmatrix}^T W^T \begin{bmatrix} \theta \\ b \end{bmatrix} \right) \geq 1 \quad \forall i \in \mathcal{D}_T \end{aligned}$$

More general problem with soft constraints and K categories :-

$$\begin{aligned} J(W, \theta_k, b_k) \quad = \quad & \frac{1}{2}||W||_F^2 + \sum_{k=1}^{K} \left[\frac{1}{2}||\theta_k||_2^2 \right. \\ & \left. + C_S \sum_{i=1}^{n_S} \mathcal{L}\left(y_i^s, \begin{bmatrix} x_i^s \\ 1 \end{bmatrix}, \begin{bmatrix} \theta_k \\ b_k \end{bmatrix} \right) + C_T \sum_{i=1}^{n_T} \mathcal{L}\left(y_i^t, W \cdot \begin{bmatrix} x_i^t \\ 1 \end{bmatrix}, \begin{bmatrix} \theta_k \\ b_k \end{bmatrix} \right) \right] \end{aligned}$$

Therefore, we refer to this method as ***Maximum Margin Domain Transform***, or ***mmdt***. The *joint optimization problem* can be formulated by adding a regularizer on Θ.

$$\begin{aligned} \min_{\boldsymbol{W},\boldsymbol{\Theta}} \quad & \frac{1}{2}||\boldsymbol{W}||_F^2 + \frac{1}{2}||\boldsymbol{\Theta}||_F^2 + \lambda\, \mathcal{L}(\boldsymbol{W}, \boldsymbol{\Theta}, \mathbf{V}, \boldsymbol{g}) \\ & + \lambda_{\mathcal{X}}\, \mathcal{L}(\boldsymbol{\Theta}, \mathbf{X}, \boldsymbol{y}) \end{aligned}$$

We perform *coordinate gradient descent* by alternating between optimizing with respect to **W** and **Θ** :

Steps :

1. Set iteration **j**= 0, $\mathbf{W}^{(j)}$= 0
2. Solve the sub-problem ($\boldsymbol{\theta}_{\mathbf{k}}^{(\mathbf{j+1})}$, $\mathbf{b}_{\mathbf{k}}^{(\mathbf{j+1})}$) =

   arg $\min_{\Theta k,\, bk}$ $\mathbf{J}(\mathbf{W}^{(j)}, \boldsymbol{\theta}_{\mathbf{k}}, \mathbf{b}_{\mathbf{k}})$ by solving:

$$\min_{\theta,b} \sum_{k=1}^{K} \left[\frac{1}{2}||\theta_k||_2^2 + C_S \sum_{i=1}^{n_S} \mathcal{L}\left(y_i^s, \begin{bmatrix} x_i^s \\ 1 \end{bmatrix}, \begin{bmatrix} \theta_k; \\ b_k \end{bmatrix} \right) + C_T \sum_{i=1}^{n_T} \mathcal{L}\left(y_i^t, W^{(j)} \cdot \begin{bmatrix} x_i^t \\ 1 \end{bmatrix}, \begin{bmatrix} \theta_k \\ b_k \end{bmatrix} \right) \right]$$

3. Solve the subproblem $W^{(j+1)}$=arg $\min_W$ $\mathbf{J}(\mathbf{W}, \boldsymbol{\theta}^{(\mathbf{j+1})}, \mathbf{b}^{(\mathbf{j+1})})$

   by solving the following equation and increment j

$$\min_{W} \quad \frac{1}{2}||W||_F^2 + C_T \sum_{k=1}^{K} \sum_{i=1}^{n_T} \mathcal{L}\left(y_i^t, W \cdot \begin{bmatrix} x_i^t \\ 1 \end{bmatrix}, \begin{bmatrix} \theta_k^{(j+1)} \\ b_k^{(j+1)} \end{bmatrix} \right)$$

4. Iterate steps 2 and 3 until convergence

Datasets

- **Amazon**
  - Part of the ***Office dataset*** and contains images from ***amazon.com*** or ***office environment images***.
  - It has images taken from ***31 categories*** with ***958 samples*** in total.

    - ***SURF BoW histogram*** features are available with vector quantized to ***800 dimension***.
- **DSLR**
    - Part of the ***Office dataset*** and contains images taken from ***DSLR camera***.
    - It has images taken from ***31 categories*** with ***157 samples*** in total.
    - ***SURF BoW histogram*** features are available with vector quantized to ***800 dimension***.
- **Webcam**
    - Part of the ***Office dataset*** and contains images taken with *varying lighting* and *pose changes* using ***a webcam***.
    - It has images taken from ***31 categories*** with ***295 samples*** in total.
    - ***SURF BoW histogram*** features are available with vector quantized to ***800 dimension***.
- **Office + Caltech 256**
    - This dataset is constructed from two datasets: **Office-31** ***(which contains 31 classes of A, W and D)*** and **Caltech-256** ***(which contains 256 classes of C).*** There are just ***10 common classes*** in both, so the **Office+Caltech dataset** is formed.
    - Total number of samples in Caltech dataset is ***1123*** and in ***Office + Caltech*** is ***2533***.

Current Progress

- Explored ***Office + Caltech*** Dataset. Four domains are included: **Caltech (C), Amazon(A), Webcam(W)** *and* **DSLR(D)**. In fact, this dataset is constructed from two datasets: ***Office-31*** (which contains ***31 classes of A, W and D***) and ***Caltech-256*** (which contains ***256 classes of C***). There are just ***10 common classes*** in both, so the Office+Caltech dataset is formed.
- Used **liblinear classifier** on this dataset.
- Implemented the complete **MMDT algorithm** by implementing both the components, the **svm based classification** and **learning the transformation matrix** for domain adaptation.
- Compared the obtained results with those presented in the paper. **Similar results** observed with our implementation.
- We tried multiple combination of source and target domains - ***Amazon, WebCam, DSLR and Caltech***. We fed them into the our implemented ***MMDT based classifier***.

- Tried different combinations of the hyperparameters - C_S (penalizes the source classification error) and C_T (penalizes the target adaptation error) and selected the model with maximum accuracy.

Results

- The dataset we used is ***Office+Caltech 256***. The ***Office dataset*** has images taken from ***3 different sources*** namely - ***Amazon (A) , WebCam (W)*** and ***DSLR (D)***. The ***Office and Caltech (C)*** dataset has ***10 common classes*** and we have computed our accuracies on these classes by varying the Source (S) and Target (T) and compared our results with the results of the proposed **MMDT** algorithm.

S - T	**MMDT (Accuracy)**	**Our Classifier (Accuracy)**
A - W	**64.6**	65.943
A - D	**56.7**	55.906
W - D	**67.0**	70.157
D - W	**74.1**	79.151
D - C	**34.1**	37.191
C - A	**49.4**	52.565

C - D	**56.5**	55.984
W - A	**47.7**	51.013
D - A	**46.9**	51.072
A - C	**36.4**	43.184
W - C	**32.2**	36.176
C - W	**63.8**	65.132

Future Approaches:

- Currently, we have explored linear transforms due to their speed and scalability, the method can be kernelized using non-linear transforms to achieve better results in some cases.
- Extending to deep features : Supervised deep convolutional neural networks (CNNs) trained on large-scale classification tasks have been shown to learn impressive mid-level structures and obtain high levels of performance on contemporary classification challenges . These models generally assume extensive training using labeled data, and testing is limited to data from the same domain.

Conclusion:

The main contributions of the paper can be summarized as follows:

- MMDT can be optimized faster than competing methods because it has fewer constraints to satisfy and because it can be optimized in linear feature space.
- Experiments show that MMDT in linear feature space outperforms competing methods in terms of multi-class accuracy even compared to some previous kernelized methods
- MMDT learns an asymmetric category independent transformation. Therefore, it can learn adaptation even when the target domain does not have any labeled examples for some categories and when the target and source features are not equivalent.
- The final iterative solution can be solved using standard QP packages, making MMDT easy to implement.

Source : amazon Target : webcam

C_s	C_t	Accuracy
0.05	1	65.717
0.05	5	64.377
0.05	0.5	65.717
0.05	2.5	64.792
0.1	1	64.453
0.1	5	64.264
0.1	0.5	63.453
0.1	2.5	64.377
0.01	1	65.038
0.01	5	63.906
0.01	0.5	65.943
0.01	2.5	64.377
0.5	1	56.000
0.5	5	61.283
0.5	0.5	51.434
0.5	2.5	60.642

best_model =>

C_s : 0.01 **C_t :** 0.5 **Accuracy :** 65.943

Source : amazon Target : dslr

C_s	C_t	Accuracy
0.05	1	55.906
0.05	5	53.858
0.05	0.5	55.551
0.05	2.5	54.409
0.1	1	55.157
0.1	5	54.449
0.1	0.5	54.488
0.1	2.5	55.118
0.01	1	53.937
0.01	5	51.890
0.01	0.5	54.961
0.01	2.5	52.520
0.5	1	50.197
0.5	5	52.874
0.5	0.5	47.677
0.5	2.5	52.244

best_model =>

C_s : 0.05 **C_t :** 1 **Accuracy :** 55.906

Source : amazon Target : caltech

C_s	C_t	Accuracy
0.05	1	40.503
0.05	5	37.781
0.05	0.5	42.145
0.05	2.5	38.591
0.1	1	41.862
0.1	5	38.843
0.1	0.5	43.184
0.1	2.5	40.091
0.01	1	36.171
0.01	5	34.318
0.01	0.5	37.649
0.01	2.5	34.895
0.5	1	42.447
0.5	5	41.080
0.5	0.5	42.425
0.5	2.5	41.743

best_model =>

C_s : 0.1 **C_t :** 0.5 **Accuracy :** 43.184

Source : webcam Target : amazon

C_s	C_t	Accuracy
0.05	1	50.216
0.05	5	49.256
0.05	0.5	51.013
0.05	2.5	49.526
0.1	1	49.353
0.1	5	48.362
0.1	0.5	49.731
0.1	2.5	48.648
0.01	1	49.246
0.01	5	48.276
0.01	0.5	50.253
0.01	2.5	48.491
0.5	1	46.202
0.5	5	45.954
0.5	0.5	45.603
0.5	2.5	46.083

best_model =>

C_s : 0.05 **C_t :** 0.5 **Accuracy :** 51.013

Source : webcam Target : dslr

C_s	C_t	Accuracy
0.05	1	66.811
0.05	5	63.504
0.05	0.5	68.701
0.05	2.5	64.370
0.1	1	67.717
0.1	5	65.512
0.1	0.5	70.157
0.1	2.5	66.417
0.01	1	59.685
0.01	5	57.598
0.01	0.5	61.654
0.01	2.5	58.583
0.5	1	68.740
0.5	5	66.339
0.5	0.5	69.488
0.5	2.5	66.772

best_model =>

C_s : 0.1 **C_t :** 0.5 **Accuracy :** 70.157

Source : webcam Target : caltech

C_s	C_t	Accuracy
0.05	1	34.245
0.05	5	32.644
0.05	0.5	35.554
0.05	2.5	33.088
0.1	1	35.256
0.1	5	33.522
0.1	0.5	36.176
0.1	2.5	34.053
0.01	1	30.645
0.01	5	29.602
0.01	0.5	31.766
0.01	2.5	29.817
0.5	1	34.849
0.5	5	33.683
0.5	0.5	35.256
0.5	2.5	34.209

best_model =>

C_s : 0.1 **C_t :** 0.5 **Accuracy :** 36.176

Source : dslr Target : amazon

C_s	C_t	Accuracy
0.05	1	50.323
0.05	5	49.321
0.05	0.5	51.072
0.05	2.5	49.520
0.1	1	50.469
0.1	5	49.558
0.1	0.5	51.061
0.1	2.5	49.747
0.01	1	47.780
0.01	5	47.047
0.01	0.5	48.658
0.01	2.5	47.214
0.5	1	48.561
0.5	5	47.953
0.5	0.5	47.877
0.5	2.5	48.211

best_model =>

C_s : 0.05 **C_t :** 0.5 **Accuracy :** 51.072

Source : dslr Target : webcam

C_s	C_t	Accuracy
0.05	1	75.887
0.05	5	73.377
0.05	0.5	77.830
0.05	2.5	74.094
0.1	1	77.887
0.1	5	75.547
0.1	0.5	79.151
0.1	2.5	76.321
0.01	1	68.453
0.01	5	66.887
0.01	0.5	70.094
0.01	2.5	67.358
0.5	1	78.038
0.5	5	76.755
0.5	0.5	78.642
0.5	2.5	77.226

best_model =>

C_s : 0.1 **C_t :** 0.5 **Accuracy :** 79.151

Source : dslr Target : caltech

C_s	C_t	Accuracy
0.05	1	35.091
0.05	5	33.527
0.05	0.5	36.555
0.05	2.5	33.980
0.1	1	36.258
0.1	5	34.378
0.1	0.5	37.191
0.1	2.5	34.913
0.01	1	31.825
0.01	5	30.773
0.01	0.5	32.832
0.01	2.5	31.057
0.5	1	35.636
0.5	5	34.657
0.5	0.5	35.851
0.5	2.5	35.114

best_model =>

C_s : 0.1 **C_t :** 0.5 **Accuracy :** 37.191

Source : caltech Target : amazon

C_s	C_t	Accuracy
0.05	1	51.859
0.05	5	50.361
0.05	0.5	52.565
0.05	2.5	50.830
0.1	1	52.069
0.1	5	50.927
0.1	0.5	52.484
0.1	2.5	51.298
0.01	1	49.025
0.01	5	47.877
0.01	0.5	49.952
0.01	2.5	48.136
0.5	1	49.617
0.5	5	49.327
0.5	0.5	48.459
0.5	2.5	49.607

best_model =>

C_s : 0.05 **C_t :** 0.5 **Accuracy :** 52.565

Source : caltech Target : webcam

C_s	C_t	Accuracy
0.05	1	64.321
0.05	5	62.226
0.05	0.5	65.132
0.05	2.5	63.019
0.1	1	63.358
0.1	5	62.094
0.1	0.5	64.226
0.1	2.5	62.679
0.01	1	61.151
0.01	5	59.792
0.01	0.5	62.547
0.01	2.5	60.189
0.5	1	56.509
0.5	5	59.132
0.5	0.5	53.264
0.5	2.5	58.679

best_model =>

C_s : 0.05 **C_t :** 0.5 **Accuracy :** 65.132

Source : caltech Target : dslr

C_s	C_t	Accuracy
0.05	1	54.252
0.05	5	51.260
0.05	0.5	55.984
0.05	2.5	52.520
0.1	1	54.882
0.1	5	51.811
0.1	0.5	55.315
0.1	2.5	52.953
0.01	1	50.591
0.01	5	48.346
0.01	0.5	52.598
0.01	2.5	48.898
0.5	1	50.787
0.5	5	51.220
0.5	0.5	49.843
0.5	2.5	51.654

best_model =>

C_s : 0.05 **C_t :** 0.5 **Accuracy :** 55.984