RWTH Aachen University
Faculty of Mathematics, Computer Science and Natural Sciences
Media Informatics

Master's Thesis

Development of a Source Library for Transfer Learning: Leveraging Clustering and Contrastive Learning Techniques

Sneha Banerjee
Matriculation Number: 430069

Thursday 28th March, 2024

1st Examiner: **Prof. Dr. Christian Bauckhage, University of Bonn**
2nd Examiner: **Prof. Dr.-Ing. Alexander Verl, ISW, University of Stuttgart**
Supervisor: **Daniel Schiller**

Declaration of Originality

Name	Sneha Banerjee
Matriculation number	430069
Address	Allmandring 8A,70569 Stuttgart
Title	*Development of a Source Library for Transfer Learning: Leveraging Clustering and Contrastive Learning Techniques*

I now declare,

- that I wrote this work independently,
- that no sources other than those stated are used and that all statements taken from other works—directly or figuratively—are marked as such,
- that the work submitted was not the subject of any other examination procedure, either in its entirety or in substantial parts,
- that I have not published the work in whole or in part, and
- that my work does not violate any rights of third parties and that I exempt the University against any claims of third parties.

Abstract

Contents

1 Introduction **1**
1.1 Motivation . . . 1
1.2 Task and Goal . . . 1
1.3 Approach . . . 1
1.4 Contribution . . . 1

2 Related Work **2**

3 Fundamentals **3**
3.1 Bin Picking . . . 3
3.1.1 Definition and Application . . . 3
3.1.2 Gripper Types . . . 3
3.1.3 Model-Based Approaches . . . 3
3.1.4 Model-Free Approaches . . . 3
3.1.5 Robot System . . . 3
3.2 Deep learning-based 3D Image processing . . . 3
3.2.1 Convolution Neural Network . . . 3
3.2.2 Autoencoders . . . 4
3.2.3 PointNet Autoencoder . . . 6
3.2.4 Siamese Network . . . 6
3.3 Clustering Algorithms . . . 8
3.3.1 Density-based Clustering . . . 9
3.3.2 Distribution-based Clustering . . . 9
3.3.3 Hierarchical-based Clustering . . . 10
3.4 Evaluation metrics for Clustering Algorithms . . . 10
3.5 Transfer learning . . . 11
3.5.1 Definition . . . 11
3.5.2 Categories . . . 12
3.5.3 Relevant Algorithms . . . 13
3.6 Network generation process with PQ-Net++ . . . 13

4 Method **14**
4.1 Suitable Algorithm and Adjustments . . . 14
4.2 Whatever new Concept we propose . . . 14
4.3 Architecture . . . 14
4.4 Source and Target Objects . . . 14
4.5 Data Generation . . . 14
4.6 Important Aspects . . . 14

5 Application of algorithms **15**

6 Experiments **16**

7 Implementation in a Bin Picking Cell **17**

8 Conclusions **18**
8.1 Discussions . . . 18
8.2 Limitations . . . 18
8.3 Future Scope . . . 18

9 Summary **19**

List of Acronyms **i**

List of Figures ii

List of Tables iii

List of Symbols iv

Bibliography v

1 Introduction

1.1 Motivation

1.2 Task and Goal

1.3 Approach

1.4 Contribution

2 Related Work

3 Fundamentals

3.1 Bin Picking

3.1.1 Definition and Application

3.1.2 Gripper Types

3.1.3 Model-Based Approaches

3.1.4 Model-Free Approaches

3.1.5 Robot System

3.2 Deep learning-based 3D Image processing

3.2.1 Convolution Neural Network

Inspired from the visual cortex of humans, convolutional neural network(Convolutional Neural Network (CNN) or ConvNet) is a type of deep neural network designed for processing data which appear in a grid like manner like images, videos, etc. It was introduced by Yann LeCun et. al in [25] in 1998. It gets it name because of the usage of a special kind of linear mathematical operation called the convolution instead of using matrix multiplication as prevalent in the pre-existing neural networks. The key components of a CNN are - Convolution layer, activation function, pooling layer, loss function, output layer. The principle building block if a CNN is the convolution layer. It consists of a number of learnable filters (kernels) which can be visualised like a cubic block. The success of CNNs can be attributed to three major concepts: sparse interactions, parameter sharing and equivariant representations.

Sparse interactions In traditional fully connected network, matrix multiplication is performed which involves a parameter matrix. The interaction between input and output units are captured by a distinct parameter in the parameter matrix. But CNNs have sparse interactions, i.e. only a subset of units or neurons in a layer is connected to a local region in the preceding layer. This is done by using kernels that are significantly smaller than the input. This implies, less number of parameters need to be stored which reduces the memory consumption and also it is computationally efficient because it has to perform fewer operations. For example, if there are m inputs and n outputs, the fully connected network would need to store m x n parameters and have a runtime of $O(m \text{ x } n)$ time complexity per input. On the other hand, if we restrict the number of connections for each unit to be k, then we would have k x n and have a runtime of $O(k \text{ x } n)$ time complexity per input, where k is quite some fold lesser in magnitude as compared to m. Moreover, since convolution layers are stacked upon one another in a deep convolution network, units in the deeper layers have a larger receptive field, because of it's indirect interaction with a larger region in the input.

Parameter sharing Another important focus behind using the convolution layer was to reduce the number of parameters in a Fully Connected Network (FCN). For example in a 1024*1024 image, a FCN would have over 1 million hidden units, which means it would have over 1 trillion trainable parameters. But the pixels in an image are only locally correlated. So, CNNs make use of the kernels to limit the focus on smaller regions on the image at a time known as the receptive field. This significantly reduces the number of trainable parameters, thus reducing the memory consumption. [6]. These layers perform a convolution operation between the input (eg. image) I and the kernel K to produce an output S known as the feature map as in Eq.1.

Equivariant Representations A function is said to be equivariant if the input to a function is changed, then the output changes in the similar way. Because of the parameter sharing mechanism, convolutions operations are translation equivariant. When the kernels the applied on an input image, the convolution layers generate a 2D map of where the particular feature occurs in the particular image. Furthermore, a pixel is related to it's neighboring pixels to form a meaningful context(create a feature e.g. an edge in an image) but it is not limited to where it can occur throughout the image. Thus it creates multiple filters

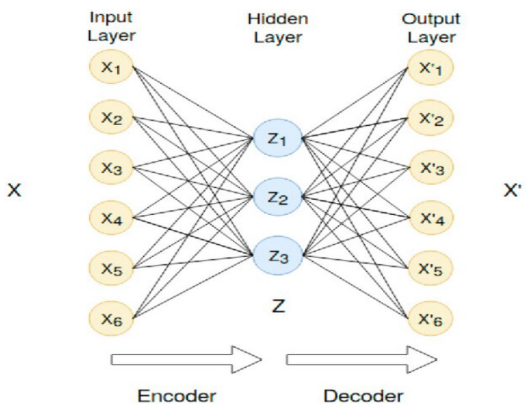

Figure 3.2.1: Architecture of an undercomplete autoencoder.[28]

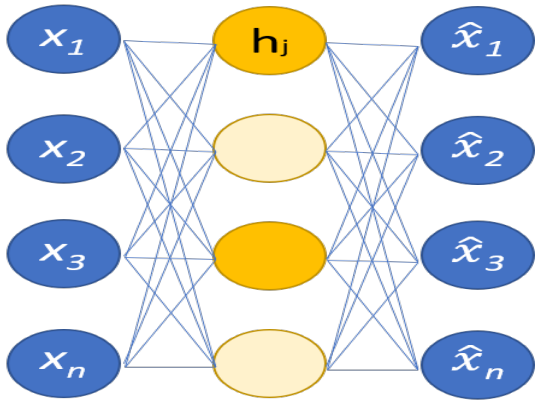

Figure 3.2.2: Architecture of a single layer sparse encoder. The hidden nodes in bright yellow are active, while the ones in light yellow are set to zero, hence inactive[2]

each of which look for the same feature throughout the image. But it is also to be kept in mind that convolution is not equivariant to other geometric and affine transformations like rotation, scaling, etc. Since convolution operations are commutative in nature, Eq. 1 can also be written as Eq.2. Typically, Eq. 2 is easier to incorporate into a machine learning library since values for both 'm' and 'n' varies within a small range.[13]

$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(m,n)K(i-m,j-n) \tag{1}$$

$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(i-m,j-n)K(m,n) \tag{2}$$

Activation function The next step in a CNN is to apply an activation function. the purpose of using a non linear activation function is to capture the non-linearity in the data. Moreover if non-linearity is not used in between the multiple layers of a neural network, the network is effectively only one layer deep which is not capable even to capture the non-linearity in real world datasets. Rectified Linear Unit (ReLU) is the most commonly used non-linear activation function used in CNNs because unlike sigmoid function or tanh function it does not penalise "too correct" data points. Another remarkable benefit of using ReLU is it's ability to propagate gradient through deep networks with a constant factor. Also it is more memory efficient to use ReLU as it doesn't require to store the ReLU outputs separately as compared to tanh outputs.

Pooling The third step of a CNN is to use the pooling layer. The main purpose of this operation is to make the detection of the features robust to the exact location of the eye (i.e. invariant to small translations). The different types of pooling operations are - max pooling and average pooling. It also helps in reducing the dimensionality of the input without losing too much information. This is done to make the computations faster down the deeper layers of the network. If the pooling operations are performed after every k pixels, then the next layer processes inputs that are k times lesser. Since the number of parameters in a layer are dependent on the size of the input to the layer, it significantly reduces the computational overhead on using pooling operations.

Complete CNN

3.2.2 Autoencoders

Autoencoders are a special type of feedforward neural network in which the output tries to reconstruct the input. It is predominantly used in unsupervised learning for the tasks if dimensionality reduction, learning feature representations, and for data compression.It tries the encode the input data into a more compact representation with lower dimensions called the "code"[13] or "latent space". The idea is that this latent space representation should capture the most vital aspects pf the input data. This is done by

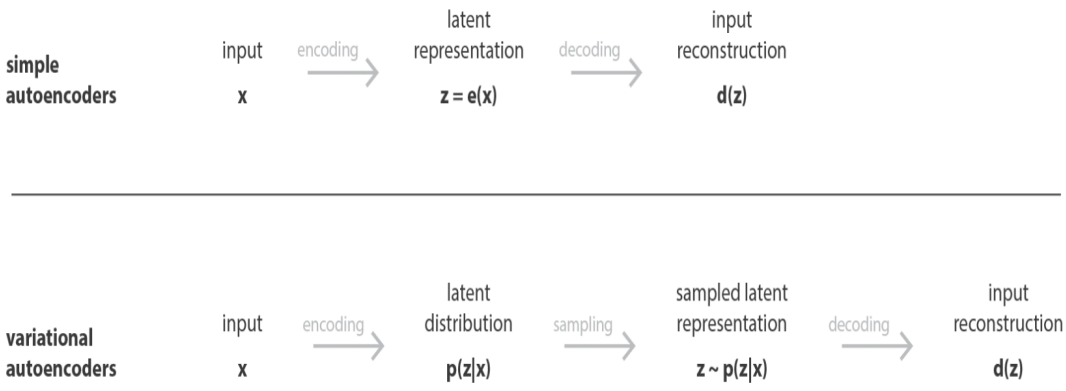

Figure 3.2.3: Difference between a vanilla autoencoder and a Variational Autoencoders (VAE).[39]

the first part of the network called the encoder. The second part of the network, the decoder then tries to decode this compressed representation of the input data to reconstruct the original input data as accurately as possible. during the training of an autoencoder, the goal is to minimise this reconstruction loss, i.e. the difference between the original input data and the output of the decoder. There are different types of autoencoders - undercomplete autoencoders, convolutional autoencoders, regularized autoencoders, and variational autoencoders,

Undercomplete autoencoders ensures that the dimension of the latent space representation is less than the dimension of the original input data as show in Fig.3.2.1. Because, the output of the decoder to be the exact copy of the input is of no use. Rather, if the dimension of the code is less than that of the input, then it ensures that the autoencoder learns those representative features of the input data which are most salient[13]. Convolutional autoencoders are an extension of traditional autoencoders where convolution layers are used as building blocks in both the encoder and the decoder part of the autoencoder. After training the network, the encoder part is used for ectracting the features of the data and the decoder part is used for the reconstruction of the input data.

Regularized autoencoders are a special type of autoencoders that employs one or more regularisation techniques to prevent the model from overfitting. The different types of regularisations used are - L1 or L2 Regularisation which adds a penalty to the loss function depending on L1 or L2 norm of the model weights As a result of this, the model is capable of learning sparse representation of the training data where many weights are set to a very small value or zero as shown in Fig3.2.2. Dropout is also used as a regularization technique where a percentage of the model's nodes are set to zero during training so that the model doesn't rely heavily on any the weight of any particular node, thus preventing overfitting and improving generalisation on unseen data. The previously mentioned regularisation techniques give rise to a variation of regularised autoencoders called the sparse autpencoders[29]. Sometimes noise is also added to the training data or the hidden layers of the model to increase the robustness the model giving rise to denoising autoencoders[40]. One more regularisation technique if to apply a contractive regularization term based on Frobenius norm of the Jacobian matrix of the model's hidden layer activations with respect to the input data[34, 2]. This ensures that a small neighborhood of the input data corresponds to a small neighborhood in the latent space representation, which means small perturbations in the input data leads to small or zero variation in the latent space representation[34, 2]. By doing so, it makes the model more robust to small changes in the input data, thus preventing overfitting.

VAEs are a distinct type of autoencoders which produces latent space representations that are continuous, which allows random sampling and interpolation for the generation of new datapoints. The difference between a vanilla autoencoder and a VAE is shown in Fig.3.2.3. Instead of producing a single vector for the latent space representation, it generates two vectors: a vector of means μ and a vector of standard deviations γ for all datapoints. An encoding is then sampled from a distribution with mean μ and standard deviation

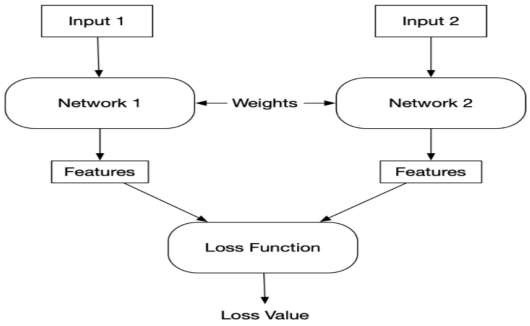

Figure 3.2.4: A generic siamese network.[1]

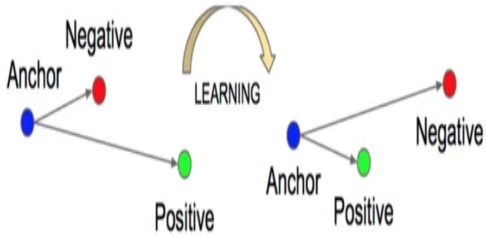

Figure 3.2.5: Before (left) and after (right) minimizing triplet loss function[38]

γ. Thus even for the same input, i.e. when the mean and the standard deviation are the same, the sampled encoding would vary because of the involvement of the sampling procedure. The mean vector controls the position where the encoding of the input should have it's centre and the standard deviation controls the area over which the sampled encoding is allowed to vary from the mean. As a result of this, the decoder learns to decode not just the encoding of a single point but also the points in the neighborhood and hence a continuous latent space representation is obtained. In order to ensure that the latent space representation satisfies that the nearby encoding are similar to each other on a local scale while also facilitating interpolation on a global scale VAEs jointly optimizes the reconstruction loss and the Kullback–Leibler divergence (KL)[24] loss.[21, 18]

3.2.3 PointNet Autoencoder

3.2.4 Siamese Network

Traditional deep learning models have shown ground breaking results in a multitude of applications like image recognition and classification, web scraping, speech recognition and caption generation. But it has a limitation that the models tend to perform well if the training and testing data points are from the same distribution. If the model is to be made capable to predict datapoints belonging to an unknown class, we need to re-train the model with an abundant number of training samples from that class and then predict it with unseen samples of that class. This technique can often be computationally expensive and can increase expotentially if the number of classes increases. Also generating a huge number of samples for a new class of objects can be tedious and quite challenging in multiple scenarios. The Siamese network[22, 4] is one such way to mitigate the above mentioned problem. The idea behind siamese networks is that two identical copies of the neural neural network with shared weights process two distinct datapoints as shown in Fig.3.2.4. Because the networks have shared parameters, it ensures that two very similar datapoints cannot have too different locations in the feature maps as both the networks evaluate the same function. In siamese networks, the model learns a similarity function, the loss function, which plays the pivotal role in determining how similar or dissimilarthe datapoints are to one another. The two main loss functions used in siamese networks are contrastive loss function[4] and triplet loss function[3]. Since the goal of a siamese network is not to perform classification task on datapoints, rather to compute the similarity between them, contrative loss functions are more suited for this task as compared to cross-entropy loss functions, as it is for classification tasks.

Contrastive loss function evaluates how capable a siamese network is in deciding the similarity between two given datapoints as is given by Eq.3[1]

$$\mathcal{L}_{\text{con}}(Y, D_w) = (1-Y)\frac{1}{2}(D_w)^2 + (Y)\frac{1}{2}max(0, m - D_w)^2 \tag{3}$$

where $\mathcal{L}_{\text{con}}$ is the calculated contrastive loss, D_w is the Euclidean distance between the outputs of the twin networks[1]. Y takes the value 0 or 1. If the two datapoints belong to the same class then Y takes the value

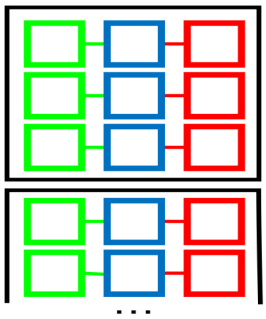

Figure 3.2.6: Minibatches in offline triplet mining. The green boxes are positive datapoints, the blue boxes are anchor datapoints and the red boxes are negative datapoints.

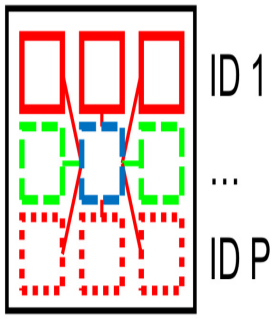

Figure 3.2.7: Triplets constructed only within the minibatch. There are P object classes in the minibatch. The green boxes are positive datapoints, the blue boxe is the anchor datapoint and the red boxes are negative datapoints.

0, otherwise it takes the value 1[1]. max() is a function to choose the higher of the two values, 0 or $m - D_w$ where m is the margin and has a value greater than 0[1]. The usage of a margin in this equation ensures that the pair of datapoints which the dissimilar beyond this margin value are excluded while calculating the loss function[1]. The euclidean distance between the embeddings of the two datapoints are given by the Eq.4[1]

$$D_w(X_1, X_2) = \sqrt[2]{\{G_w(X_1) - G_w(X_2)\}^2} \tag{4}$$

where $G_w(X_i)$ is the output of the twin network such that $i \in \{1,2\}$ as show in Fig.3.2.4. **Triplet loss** function[42] trains the siamese network in a way that a triplet of datapoints are presented to the siamese network. by applying triplet loss the model tries to group the datapoints similar to the anchor datapoint, close to one another while increasing the distance between the anchor datapoint and the datapoints dissimilar to it as shown in Fig.3.2.5. In order to do so, a positive value called margin is used which increases the distance between dissimilar datapoints and also eliminates the output of any trivial solution. Euclidean distance or cosine distance is used as the distance metric while calculating the triplet loss and shown in Eq.5[15, 1]

$$\mathcal{L}_{\text{tri}}(a,p,n) = \sum_{\substack{a,p,n \\ y_a = y_p != y_n}} max(0, d(a,p) - d(a,n) + margin) \tag{5}$$

where y_x is the class label for the datapoint x for $x \in \{a,p,n\}$, a is the anchor datapoint, p is the positive example having the same class label as the anchor datapoint and n is the negative example having a different class label as compared to the anchor datapoint. but there are some practical problems related to the use of triplet loss function. The number of possible triplets grow cubically with the increase in the size of the dataset. Also most of the triplets are often uninformative. Thus mining "hard triplets" or the ones crucial for the learning of the model becomes a challenging task. As per the definition of the triplet loss function, the triplets can be of three types: easy triplets, hard triplets and medium hard triplets. The triplets which satisfy the condition $d(a,p) + margin < d(a,n)$ results to the loss value being 0, hence are called the easy triplets.[38] For the triplet for which the negative datapoint is closer to the anchor as compared to the positive datapoint i.e. $d(a,n) < d(a,p)$ are the examples of hard triplets.[38] And the medium hard triplets are the ones where the negative datapoints are not closer to the anchor datapoint as compared to the positive datapoint but still results in positive loss i.e. $d(a,p) < d(a,n) < d(a,p) + margin$.[38] Thus the medium hard triplets are the most essential for the model training as they constitute for the most useful information. Two triplet mining procedures are used to mitigate the issue of excessive computational overhead with increase in dataset size. They are offline triplet mining and online triplet mining. In offline hard triplet mining, at first the dataset is manually processed to find hard triplets and then they are used for training the model. But manually finding the hard triplets can be quite tedious. Online triplet mining is a better approach as compared to the one mentioned above. The model is trained in minibatches of the dataset as shown in Fig.3.2.6. Using only the triplets that were mined from the dataset could be a wasteful design

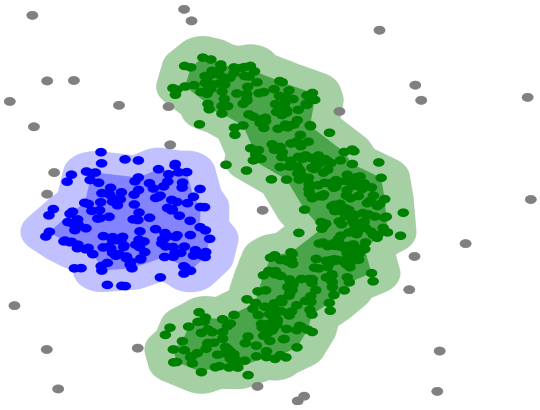

Figure 3.3.1: Density-Based Spatial Clustering of Applications with Noise (DBSCAN) can find non-linearly separable non-convex clusters.[9]

choice in such scenario. So in a minibatch, it containes way more potential triplets than the ones that were mined. So each member of another triplet becomes an additional negative datapoint example. But both hard positive and hard negative datapoints are required for training. An even better design to mitigate that would be to choose let's say K datapoints from P different classes in the dataset, where the K positive examples serve as hard positives while the rest of the datapoints in the minibatch serve as hard negatives as shown in Fig.3.2.7[15].

Thus siamese networks are very robust to the class imbalance in dataset which is very common in real datasets. Since it depends on very few datapoints of a particular object class, it has good generalisation capabilities for unseen datapoints in the future. They are also very robust to small perturbationsin the datapoints due to difference in lighting conditions or orientation or background, as they keep similar objects close to each other even if the datapoints are slightly different. Since the siamese networks places similar datapoints together while increasing the distance between dissimilar datapoints, they are often useful for learning semantic similarity within the datapoints. However despite it's benefit, training a siamese network takes longer as compared to traditional neural networks because it involves learning from quadratic pair of datapoints.[1]

3.3 Clustering Algorithms

When the dataset does not have any labelled data, then it is said to be unsupervised learning. In this case, there is no ground truth available to measure the correctness of the outputs generated by the Machine Learning (ML) models. The primary focus of unsupervised learning is to find hidden and interesting pattern in the data. Unsupervised learning is of utmost importance in the Artificial Intelligence (AI) world as several real world datasets do not have available annotations, which requires a lot of human effort. Unsupervised learning algortihms can be broadly categorized into the following domains-clustering, dimensionality reduction, and association analysis. Clustering algorithms aim at grouping unlabelled data into groups or clusters based on how similar or dissimilar the datapoints are to one another. It can reveil underlaying hidden pattern in the data and is used in applications like image segmentation, fraud detection, etc. Dimensionality reduction is reduces the number of irrelavant features or dimensions of the dataset. The inclusion of more features does help in the better representation of the dataset but it also significantly increases the memory consumption and complexity to work with it. Also it is often difficult to visualise real-life datasets with too many features. Association analysisis a rule-based unsupervised learning method that reveals the relationship between attributes in the dataset. It is used in applications like market analysis, intrusion detection, etc.Clustering algorithms can be broadly classified into the following categories: density-based, distribution-based, and hierarchical-based.

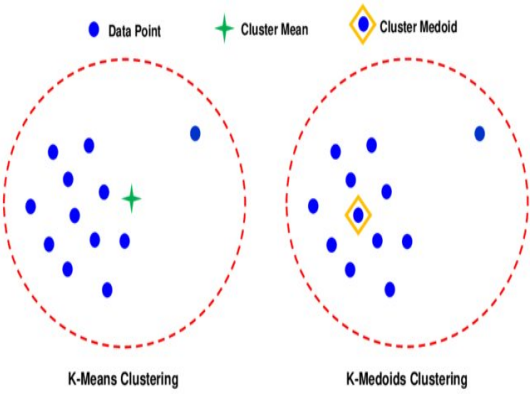

Figure 3.3.2: Difference between K-Means and K-Medoids.[11]

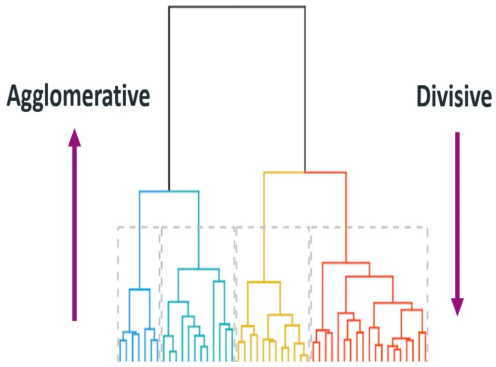

Figure 3.3.3: A dendogram in hierarchical clustering.[16]

3.3.1 Density-based Clustering

In density-based clustering, the algorithm looks for areas of high concentration of the datapoints and groups them as a cluster. The benefit of this algorithm is that the shape a cluster can be is not limited and hence doesn't necessarily have to be convex in nature as show in Fig. 3.3.1. Moreover, these algorithms are also very robust to outliers as they do not force the outliers to belong to any category but are rather ignored as it is unlikely that outliers can form an area of high concentration of datapoints. I have carried on the experiments on this thesis on two such density-based clustering algortihms called DBSCAN[12] and Hierarchical Density-Based Spatial Clustering of Applications with Noise (HDBSCAN) algorithms[27]. The available scikit-learn implementations were used for this purpose[32]. The DBSCAN[12] algortihm does not require the users to define the number of clusters to be generated which doesn't force dissimilar datapoints to belong to the same cluster. However, this algorithm was still sensitive to two parameters ε, the maximum distance betwwen the datapoints to be considered in the same cluster and the minimum number of samples in the cluster which the user needs to define [32]. But finding an optimum value for these parameters often require domain expertise and are dependent on the data. The HDBSCAN[27] algorithm mitigates this issue and thus does not require the user to define these two parameters. It is a hierarchical density-based clustering that performs the DBSCAN algorithm over multiple ε values to find the most stable result.

3.3.2 Distribution-based Clustering

In distribution-based algorithms, a datapoint is said to be a member of the cluster depending on the probability of it's membership to the cluster. The more the distance of a point increases from the centre of the cluster, the less is it's probability of belonging to that cluster. Centroid-based algorithms groups the datapoints based on some initial cluster centres. Once all the dataspoints are softly assigned to some cluster membership, the cluster centres are recalculated and this process is iterated until convergence. These algorithms are sensitive to the initial parameters like the cluster centres chosen in the first step. Another major disadvantage of these clustering algorithms are that they always form spherical clusters. The user also needs to define the number of clusters the dataset is to be grouped in, which makes it sensitive to outliers. However, these algorithms can be executed very fast and we have used two such algorithms in our experiments- K-Means[19]and K-Medoids[20]. In K-Means, the mean of the datapoints of the clusters is assigned as the cluster-centroid. It might not be an actual datapoint in the dataset, rather a blurred, noisy average of a datapoints in the cluster. On the contrary, the K-Medoids algortihm assigns an actual datapoint of the dataset, that is most centrally located as the cluster centroid as shown in Fig.3.3.2. Thus K-Medoids is more robust to outliers and noises as compared to K-Means.

3.3.3 Hierarchical-based Clustering

Hierarchical-based clustering algorithm that form a hierarchy of clusters. Datapoints in a cluster are more similar to each other as compared to other groups. this hierarchy of clusters is visualised by a hierarchy tree called dendograms. hierarchical clustering algorithms can be of two types - agglomerative and divisive. In agglomerative clustering, each datapoint is considered as a separate cluster in the first step. Then these clusters are merged into one another until only one cluster remains. Thus at the end, the last level cluster consists of all teh datapoints in the dataset. The divisive method is the reverse procedure of the aggomerative method. In the beginning, all the datapoints are considered to be in a single cluster and gradually the cluster is broken into smaller clusters, untill each cluster consists of only one datapoint. A visual representation of a dendogram has been shown in Fig.3.3.3

3.4 Evaluation metrics for Clustering Algorithms

The ground truth or labels are available in supervised learning. So to determine the quality of the clustering algorithm we can calculate the level of accuracy by comparing the labels predicted by a model to the ground truth. But most real world datasets do not have ground truth. Therefore, it makes it challenging in unsupervised learning paradigm to determine the quality of the clustering algorithm. But a good clustering algorithm is still expected to have little difference between the objects in a cluster i.e. small within-cluster variance and more difference between objects from different clusters i.e. large between-cluster variance. Thus the evaluation metrics used in the literature could be categorised into two broad groups: extrinsic measures and intrinsic measures, depending on the availability of the ground truth. Extrinsic measures, when the ground truth is available uses metrices like Rand Index, Mutual Information, V-measure and Fowlkes-Mallows Scores. Intrinsic measures, when the ground truth is not available mostly uses Silhouette Score, Calinski Harabaz Index and David Bouldin index.

Silhouette Score is used to calculate the distance between points within a cluster as compared to it's distance from the points in the neighbouring cluster and is given by Eq. 6

$$S_c = \frac{(n_c - i_c)}{max(n_c, i_c)} \tag{6}$$

where S_c is the Silhouette Coefficient, n_c is the average of the nearest-cluster distance of each datapoint and i_c is the average distance between the datapoint within the cluster. The Silhouette coefficient lies within the range $[-1, 1]$, where the the more closer the coefficient is to 1, the better is the clustering results. When the Silhouette Coefficient is close to 0, it implies that the datapoints are on or very close to the decision boundary between adjacent clusters, whereas negative values imply that the datapoints have been wrongly classified. But the drawbacks of using this metric is that is tends to favour dense and convex clusters. Since it takes into account, the average of teh distance within a cluster as well as between adjacent clusters, it doesn't perform well different clusters have a significant difference in their respective sizes, which is quite a common phenomenon in real world datasets. The chosen distance metric for the calculation of the Silhouette score also has a significant effect on it and can lead to very different results.

Calinski-Harabasz Index(C-H Index) or the Variance Ratio Criterion is used to measure the dispersion between the datapoints within a cluster as compared to in-between clusters as in given by Eq. 7

$$S_{CH} = \frac{tr(B_k)}{tr(W_k)} \times \frac{(n_E - k)}{k - 1} \tag{7}$$

where S_{CH} is the C-H Index, $tr(B_k)$ is the trace of the in-between cluster dispersion matrix is in Eq. 8 and $tr(W_k)$ is the trace of the within-cluster dispersion matrix as in Eq. 9

$$B_k = \sum_{q=1}^{k} n_q (c_q - c_o)(c_q - c_o)^T \tag{8}$$

$$W_k = \sum_{q=1}^{k} \sum_{x \in C_q} (x - c_q)(x - c_q)^T \tag{9}$$

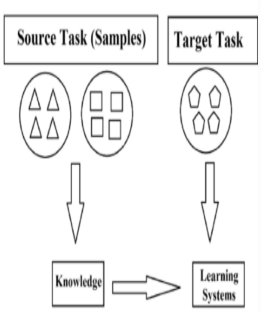

Figure 3.5.1: Model trained for a task in the similar domain is re-used for a new task.[17]

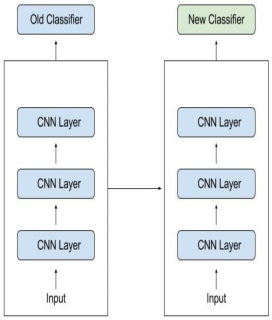

Figure 3.5.2: A model trained on a different task is adapted to be used for a new task.[44]

where c_q is the centroid of the cluster q and c_o is the overall centroid of the datapoints. B_k calculates the quality of the separation between two clusters, i.e., higher the value of B_k, better it is. W_k calculates the distance of a datapoint (s) from the centroid of the cluster, and thus measures the cohesiveness or compactness of the datapoints within a cluster, i.e, smaller the value of W_k, better it is. Distribution-based clustering algorithms like K-Means tries to minimise the value of W_k. Thus according to the equation of C-H Index in Eq. 7, the higher the C-H index, the better is the clustering results. Because sum-of squared Euclidean distance is used to calculate the distance of the datapoints from their respective cluster centroids, C-H Index tends to favour convex clusters, which is a major drawback of using this metric as in Silhouette Coefficient.[5]

3.5 Transfer learning

Transfer learning is an optimization in machine learning techniques where a model trained for a particular task let's say task 1 is used for another task, task 2. It uses the knowledge learnt by the model in the previous task to be adapted for the new task. Trasnfer learning has it's benefits in a multitude of applications because real world problems doesn't necessarily always have an abundant amount of labelled data which can beused to train a model from scratch[43]. It can be used for tasks which intend to solve different but related problems[43]. In other words, if the domain of the data for task 2 is similar to that of task 1, transfer learning can be used as shown in Fig.3.5.1.Several approaches have been adapted to use transfer learning. Let's say we want to solve task T_1, but we don't have adequate data to train a deep neural network for it. But we have ample data to solve a similar task T_2. So we can train a model for task T_2 and then use it for task T_1. We might need to re-train only the later layers or all the layers of the model, but that heavily depends on the problem at hand. Another approach could be to use an existing pre-trained network. There are a lot a available open-source models which can be re-trained and re-used depending on the problem. Some of the popular models that are used as pre-trained models in transfer learning are Oxford VGG Model[36],Google Inception model[37], Microsoft ResNet model[14]. Another approach could be to use transfer learning for the purpose of feature extraction. Transfer learning has found a lot of application in the domain of computer vision and natural language processing because it requires a lot of data for training such complex models from scratch. In the domain of computer vision, we know that the early layers of a deep neural network focuses on learning the low level features like the detection of edges, while the middle layers focus on learning the mid level features like the detection of shapes. In transfer learning, these pre-trained early and middle layers could be used verbatim and only the latter layers which focus on the high level features could be re-trained for the new task as shown in fig3.5.2. Transfer learning has numerous advantages like significant reduction in the amount of training time[26], improvement in performance for some neural networks[31] and less reliance on a huge amount of data[10, 23, 46].

3.5.1 Definition

According to [43, 30], transfer learning can be formally defined as the following. A domain $\mathcal{D}$ of the input data, with the feature space $\mathcal{X}$ and a marginal probability distribution $P(X)$ such that $X = \{x_1, ..., x_n\} \in \mathcal{X}$,

Transfer learning Settings	Related areas	Source Domain Labels	Target Domain Labels	Tasks
Inductive Transfer Learning	Multi-task learning	Available	Available	Regression, Classification
	Self-taught Learning	Unavailable	Available	
Transductive Transfer Learning	Domain Adaptation, Simple Selection bias, Co-variate Shift	Available	Unavailable	
Unsupervised Transfer Learning		Unavailable	Unavailable	Clustering, Dimensionality Reduction

Table 1: Different settings of transfer learning [30]

where n is the number of feature vectors in X. In the given input data domain $\mathcal{D}$, the task $\mathcal{T}$ has a label space $\mathcal{Y}$ and a predictive function $f(\cdot)$ which is learned from the feature vector and label pairs (x_i, y_i) such that $x_i \in \mathcal{X}$ and $y_i \in \mathcal{Y}$ and $f(x)$ is the learner that predicts the label value for x. As per the above definition, the source domain is defined as $\mathcal{D}_\mathcal{S}$ and the corresponding source task as $\mathcal{T}_\mathcal{S}$.The target domain is defined as $\mathcal{D}_\mathcal{T}$ and the corresponding target task as $\mathcal{T}_\mathcal{T}$. Transfer learning aims at improving the target predictive function $f_\mathcal{T}(\cdot)$ by using the related information from $\mathcal{D}_\mathcal{S}$ and $\mathcal{T}_\mathcal{S}$ where $\mathcal{D}_\mathcal{S} \neq \mathcal{D}_\mathcal{T}$ or $\mathcal{T}_\mathcal{S} \neq \mathcal{T}_\mathcal{T}$. Since $\mathcal{D}_\mathcal{S} = \{\mathcal{X}_\mathcal{S}, P(X_\mathcal{S})\}$ and $\mathcal{D}_\mathcal{T} = \{\mathcal{X}_\mathcal{T}, P(X_\mathcal{T})\}$, here the condition $\mathcal{D}_\mathcal{S} \neq \mathcal{D}_\mathcal{T}$ means that $\mathcal{X}_\mathcal{S} \neq \mathcal{X}_\mathcal{T}$ and/or $P(X_\mathcal{S}) \neq P(X_\mathcal{T})$.The scenario where $\mathcal{X}_\mathcal{S} \neq \mathcal{X}_\mathcal{T}$ in the context of transfer learning is defined as heterogeneous transfer learning. The scenario where $\mathcal{X}_\mathcal{S} = \mathcal{X}_\mathcal{T}$ is defined as homogeneous transfer learning. The scenario where $P(X_\mathcal{S}) \neq P(X_\mathcal{T})$ means the source and the target data are from different domains, i.e., they belong to different marginal distributions. Transfer learning algortihms are not expected to give optimal results in ths case. Refering back to the definition of transfer learning, when $\mathcal{T}_\mathcal{S} \neq \mathcal{T}_\mathcal{T}$ and it was defined that $\mathcal{T} = \{(Y), P(Y|X)\}$. this scenario is possible when $\mathcal{Y}_\mathcal{S} \neq \mathcal{Y}_\mathcal{T}$ and/or $P(Y_\mathcal{S}|X_\mathcal{S}) \neq P(Y_\mathcal{T}|X_\mathcal{T})$. The scenario where $P(Y_\mathcal{S}|X_\mathcal{S}) \neq P(Y_\mathcal{T}|X_\mathcal{T})$ implies that the conditional probability distributions between the source and target domains are different. The case $\mathcal{Y}_\mathcal{S} \neq \mathcal{Y}_\mathcal{T}$ mean the label space of the source and target domain are different.[43]

3.5.2 Categories

As per the definition of transfer learning mentioned above, it can be broadly categorized into 3 groups: inductive transfer learning, transductive transfer learning and unsupervised transfer learning as mentioned in [30]. Depending on the availability of labels of the source and target tasks, these 3 categories are possible as shown in Table 1.

Regardless of whether the source and target domains are the same or not, the target task in an inductive transfer learning situation is distinct from the source task. In this instance, in order to generate an objective predictive function $f_\mathcal{T}(\cdot)$ for usage in the target domain, some labeled data from the target domain are needed. Furthermore, inductive transfer learning can be broken down into two groups depending on the availability of source and target domain labels. When a lot of labeled data is available for the source domain, it is similar to multi-task learning. But the difference lies in the fact that the goal of inductive transfer learning is to attain good results inthe target task only by using the knowledge obtained from the source task while multi-task learning endeavors to learn both the target and source tasks concurrently[30]. If the labels pf the source data are not available, then it is similar to the self-taught learning as proposed by [33]. In such a scenario, the label space of the source data and the target data can vary which means that the knowledge obtained from the source domain cannot be utilised directly. This implies that basically the labels of the source domain are not available.[30]

On the other hand, in transductive transfer learning, the target task and the source task are same, but the sou8rce domain and the target domain are different. Depending on the similarity of the feature space and the marginal probability distribution between the source and the target domain, tranductive transfer learning too can be classfied into two groups. One scenario can be that the feature space of the source domain and

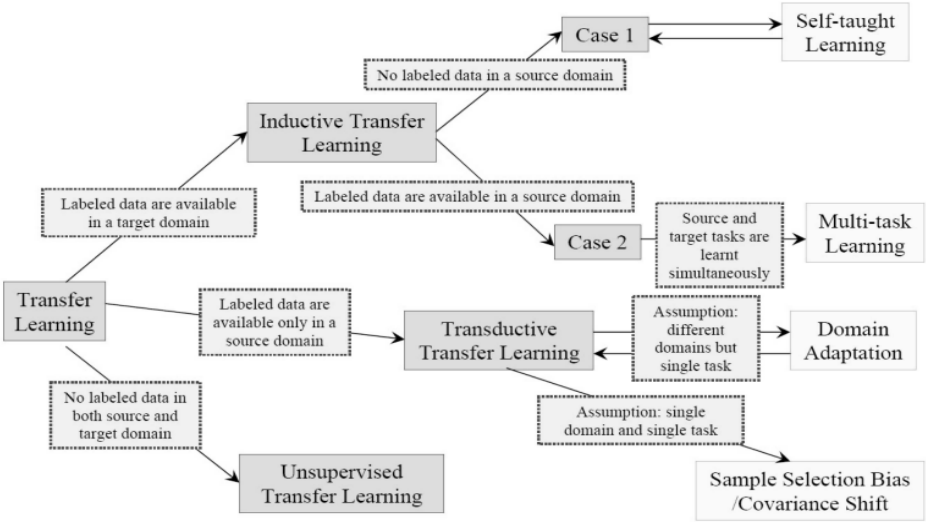

Figure 3.5.3: Overview of the different types of transfer learning.[30]

that of the target domain are different i.e. $\mathcal{X}_S \neq \mathcal{X}_T$. Again, the feature space of the two domain can be same, but the input data's marginal probability distributions are different i.e. $P(X_S) \neq P(X_T)$. Similar assumptions underlie the latter instance of the transductive transfer learning, which is related to sample selection bias [45], covariate shift [35], and domain adaptation for knowledge transfer in text categorization [8, 45].[30] Lastly, in unsupervised transfer learning, the target task is different from the source task as it was in inductive transfer learning, but the labels of neither the source domain nor the target domain is available. the goal of unsupervised transfer learning is to solve unsupervised learning problems in the target domain like density estimation, dimensionality reduction and clustering[7, 41]. The categorisation of transfer learning has been summarised in Fig.3.5.3

3.5.3 Relevant Algorithms

3.6 Network generation process with PQ-Net++

4 Method

4.1 Suitable Algorithm and Adjustments

4.2 Whatever new Concept we propose

4.3 Architecture

4.4 Source and Target Objects

4.5 Data Generation

4.6 Important Aspects

5 Application of algorithms

6 Experiments

7 Implementation in a Bin Picking Cell

8 Conclusions

8.1 Discussions

8.2 Limitations

8.3 Future Scope

9 Summary

List of Acronyms

CNN Convolutional Neural Network

FCN Fully Connected Network

ReLU Rectified Linear Unit

ML Machine Learning

AI Artificial Intelligence

DBSCAN Density-Based Spatial Clustering of Applications with Noise

HDBSCAN Hierarchical Density-Based Spatial Clustering of Applications with Noise

VAE Variational Autoencoders

KL Kullback–Leibler divergence

List of Figures

3.2.1 Architecture of an undercomplete autoencoder.[28] 4
3.2.2 Architecture of a single layer sparse encoder. The hidden nodes in bright yellow are active, while the ones in light yellow are set to zero, hence inactive[2] 4
3.2.3 Difference between a vanilla autoencoder and a VAE.[39] 5
3.2.4 A generic siamese network.[1] 6
3.2.5 Before (left) and after (right) minimizing triplet loss function[38] 6
3.2.6 Minibatches in offline triplet mining. The green boxes are positive datapoints, the blue boxes are anchor datapoints and the red boxes are negative datapoints. 7
3.2.7 Triplets constructed only within the minibatch. There are P object classes in the minibatch. The green boxes are positive datapoints, the blue boxe is the anchor datapoint and the red boxes are negative datapoints. 7
3.3.1 DBSCAN can find non-linearly separable non-convex clusters.[9] 8
3.3.2 Difference between K-Means and K-Medoids.[11] 9
3.3.3 A dendogram in hierarchical clustering.[16] 9
3.5.1 Model trained for a task in the similar domain is re-used for a new task.[17] 11
3.5.2 A model trained on a different task is adapted to be used for a new task.[44] 11
3.5.3 Overview of the different types of transfer learning.[30] 13

List of Tables

1 Different settings of transfer learning [30] 12

List of Symbols

Bibliography

[1] *A Comprehensive Guide to Siamese Neural Networks*. https://medium.com/@rinkinag24/a-comprehensive-guide-to-siamese-neural-networks-3358658c0513.

[2] *Autoencoder*. https://en.wikipedia.org/wiki/Autoencoder.

[3] Vassileios Balntas et al. "Learning local feature descriptors with triplets and shallow convolutional neural networks." In: *Bmvc*. Vol. 1. 2. 2016, p. 3.

[4] Jane Bromley et al. "Signature verification using a" siamese" time delay neural network". In: *Advances in neural information processing systems* 6 (1993).

[5] *Calinski-Harabasz Index*. https://en.wikipedia.org/wiki/Calinski-Harabasz_index.

[6] *Convolutional Neural Networks, Explained*. https://towardsdatascience.com/convolutional-neural-networks-explained-9cc5188c4939.

[7] Wenyuan Dai et al. "Self-taught clustering". In: *Proceedings of the 25th international conference on Machine learning*. 2008, pp. 200–207.

[8] Hal Daume III and Daniel Marcu. "Domain adaptation for statistical classifiers". In: *Journal of artificial Intelligence research* 26 (2006), pp. 101–126.

[9] *DBSCAN Wikipedia*. https://en.wikipedia.org/wiki/DBSCAN.

[10] Lixin Duan, Dong Xu, and Ivor Tsang. "Learning with augmented features for heterogeneous domain adaptation". In: *arXiv preprint arXiv:1206.4660* (2012).

[11] Alireza Entezami, Hassan Sarmadi, and Behzad Saeedi Razavi. "An innovative hybrid strategy for structural health monitoring by modal flexibility and clustering methods". In: *Journal of Civil Structural Health Monitoring* 10.5 (2020), pp. 845–859.

[12] Martin Ester et al. "A density-based algorithm for discovering clusters in large spatial databases with noise". In: *kdd*. Vol. 96. 34. 1996, pp. 226–231.

[13] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016.

[14] Kaiming He et al. "Deep residual learning for image recognition". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 770–778.

[15] Alexander Hermans, Lucas Beyer, and Bastian Leibe. "In defense of the triplet loss for person re-identification". In: *arXiv preprint arXiv:1703.07737* (2017).

[16] *Hierarchical Clustering*. https://harshsharma1091996.medium.com/hierarchical-clustering-996745fe656b.

[17] Asmaul Hosna et al. "Transfer learning: a friendly introduction". In: *Journal of Big Data* 9.1 (2022), p. 102.

[18] *Intuitively Understanding Variational Autoencoders*. https://towardsdatascience.com/intuitively-understanding-variational-autoencoders-1bfe67eb5daf.

[19] Xin Jin and Jiawei Han. "K-Means Clustering". In: *Encyclopedia of Machine Learning*. Ed. by Claude Sammut and Geoffrey I. Webb. Boston, MA: Springer US, 2010, pp. 563–564. ISBN: 978-0-387-30164-8. DOI: 10.1007/978-0-387-30164-8_425. URL: https://doi.org/10.1007/978-0-387-30164-8_425.

[20] Xin Jin and Jiawei Han. "K-Medoids Clustering". In: *Encyclopedia of Machine Learning*. Ed. by Claude Sammut and Geoffrey I. Webb. Boston, MA: Springer US, 2010, pp. 564–565. ISBN: 978-0-387-30164-8. DOI: 10.1007/978-0-387-30164-8_426. URL: https://doi.org/10.1007/978-0-387-30164-8_426.

[21] Diederik P Kingma, Max Welling, et al. "An introduction to variational autoencoders". In: *Foundations and Trends® in Machine Learning* 12.4 (2019), pp. 307–392.

[22] Gregory Koch, Richard Zemel, Ruslan Salakhutdinov, et al. "Siamese neural networks for one-shot image recognition". In: *ICML deep learning workshop*. Vol. 2. 1. Lille. 2015.

[23] Brian Kulis, Kate Saenko, and Trevor Darrell. "What you saw is not what you get: Domain adaptation using asymmetric kernel transforms". In: *CVPR 2011*. IEEE. 2011, pp. 1785–1792.

[24] Solomon Kullback and Richard A Leibler. "On information and sufficiency". In: *The annals of mathematical statistics* 22.1 (1951), pp. 79–86.

[25] Yann LeCun et al. "Gradient-based learning applied to document recognition". In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324.

[26] Tong Liu et al. "Exploring transfer learning to reduce training overhead of hpc data in machine learning". In: *2019 IEEE International Conference on Networking, Architecture and Storage (NAS)*. IEEE. 2019, pp. 1–7.

[27] Claudia Malzer and Marcus Baum. "A hybrid approach to hierarchical density-based cluster selection". In: *2020 IEEE international conference on multisensor fusion and integration for intelligent systems (MFI)*. IEEE. 2020, pp. 223–228.

[28] Sumit Misra et al. "An autoencoder based model for detecting fraudulent credit card transaction". In: *Procedia Computer Science* 167 (2020), pp. 254–262.

[29] Andrew Ng et al. "Sparse autoencoder". In: *CS294A Lecture notes* 72.2011 (2011), pp. 1–19.

[30] Sinno Jialin Pan and Qiang Yang. "A survey on transfer learning". In: *IEEE Transactions on knowledge and data engineering* 22.10 (2009), pp. 1345–1359.

[31] Weike Pan et al. "Transfer learning to predict missing ratings via heterogeneous user feedbacks". In: *Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence, Barcelona, Catalonia, Spain*. 2011, p. 2318.

[32] F. Pedregosa et al. "Scikit-learn: Machine Learning in Python". In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

[33] Rajat Raina et al. "Self-taught learning: transfer learning from unlabeled data". In: *Proceedings of the 24th international conference on Machine learning*. 2007, pp. 759–766.

[34] Salah Rifai et al. "Contractive auto-encoders: Explicit invariance during feature extraction". In: *Proceedings of the 28th international conference on international conference on machine learning*. 2011, pp. 833–840.

[35] Hidetoshi Shimodaira. "Improving predictive inference under covariate shift by weighting the log-likelihood function". In: *Journal of statistical planning and inference* 90.2 (2000), pp. 227–244.

[36] Karen Simonyan and Andrew Zisserman. "Very deep convolutional networks for large-scale image recognition". In: *arXiv preprint arXiv:1409.1556* (2014).

[37] Christian Szegedy et al. "Going deeper with convolutions". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015, pp. 1–9.

[38] *Triplet Loss and Siamese Neural Networks*. `https://medium.com/@enoshshr/triplet-loss-and-siamese-neural-networks-5d363fdeba9b`.

[39] *Understanding Variational Autoencoders(VAEs)*. `https://towardsdatascience.com/understanding-variational-autoencoders-vaes-f70510919f73`.

[40] Pascal Vincent et al. "Extracting and composing robust features with denoising autoencoders". In: *Proceedings of the 25th international conference on Machine learning*. 2008, pp. 1096–1103.

[41] Zheng Wang, Yangqiu Song, and Changshui Zhang. "Transferred dimensionality reduction". In: *Machine Learning and Knowledge Discovery in Databases: European Conference, ECML PKDD 2008, Antwerp, Belgium, September 15-19, 2008, Proceedings, Part II 19*. Springer. 2008, pp. 550–565.

[42] Kilian Q Weinberger and Lawrence K Saul. "Distance metric learning for large margin nearest neighbor classification." In: *Journal of machine learning research* 10.2 (2009).

[43] Karl Weiss, Taghi M Khoshgoftaar, and DingDing Wang. "A survey of transfer learning". In: *Journal of Big data* 3.1 (2016), pp. 1–40.

[44] *What Is Transfer Learning? Exploring the Popular Deep Learning Approach*. `https://builtin.com/data-science/transfer-learning`.

[45] Bianca Zadrozny. “Learning and evaluating classifiers under sample selection bias”. In: *Proceedings of the twenty-first international conference on Machine learning*. 2004, p. 114.

[46] Yin Zhu et al. “Heterogeneous transfer learning for image classification”. In: *Proceedings of the AAAI conference on artificial intelligence*. Vol. 25. 1. 2011, pp. 1304–1309.