Appendix
Image2Point: 3D Point-Cloud Understanding with 2D Image Pretrained Models

Chenfeng Xu[1]*, Shijia Yang[1]*, Tomer Galanti[2], Bichen Wu[3], Xiangyu Yue[1], Bohan Zhai[1], Wei Zhan[1], Peter Vajda[3], Kurt Keutzer[1], and Masayoshi Tomizuka[1]

[1] University of California, Berkeley
[2] Massachusetts Institute of Technology
[3] Meta Reality Labs

A Implementation Details

Datasets. Our experiments are conducted on ModelNet 3D Warehouse, S3DIS, and SemanticKITTI datasets. For the ModelNet 3D Warehouse dataset, we train all models on the train set and evaluate on the validation set. For the S3DIS, we train all models on area 1, 2, 3, 4, 6 and evaluate on area 5. For the SemanticKITTI dataset, we train all models on splits 00-10 except 08 which is used for evaluation. For each of the datasets, all ResNet series models use the same training scheme, and all experiments are implemented with PyTorch.

Training on ModelNet 3D Warehouse dataset. In this case, coordinates of point-clouds are randomly scaled, translated, and jittered. We employ the SGD optimizer with momentum 0.9, weight-decay 10^{-4}, and initial learning rate 0.1 with cosine learning rate scheduler. Each mini batch is set to 32, and models are trained for 300 epochs. For both training and inference phase, we only utilize x, y, z coordinates without other features and set the voxel size to be 0.05. The experiments for ModelNet 3D Warehouse are all conducted on a Titan RTX GPU.

Training on the S3DIS dataset. In this case, we concatenate all subparts of an indoor scene to train and validate on. Along x, y directions, scenes are applied horizontal flip randomly. RGB features are randomly jittered, translated, and auto contrasted. Finally, we normalize and clip point-clouds. We set voxel size to 0.05, use SGD optimizer with momentum 0.9, weight-decay 10^{-4}, and initialize learning rate to 0.1 with polynomial learning rate scheduler. Each mini batch is set to 3, and models are trained for 400 epochs on 2 Titan RTX GPUs.

* Equal contribution

Training on the SemanticKITTI dataset. In this case, coordinates of each point-cloud are randomly scaled and rotated. We use SGD optimizer with momentum 0.9, weight-decay 10^{-4}, and initial learning rate 0.24 with cosine warmup learning rate scheduler. Each mini batch is set to 2, and models are trained for 15 epochs on 4 Titan RTX GPUs. For both training and inference phases, we utilize x, y, z coordinates as well as intensity feature and set voxel size to 0.05.

Most of our pretrained models were taken from open-sources [4][5][6][7][8][9], so we do not need to take time and computational resources for pretraining. We use torchsparse[10] to produce sparse 3D convolutions.

Details on Section 4.1. In this section, we take the ResNet architecture, inflate the pretrained models of different image datasets, and add linear input and output layers as shown in Section B.7. The ResNet50 was pretrained on ImageNet1K and is taken from the original PyTorch example. We use the same training recipe provided by PyTorch to train the ResNet50 on Tiny-ImageNet. The pretrained ResNet50 on ImageNet21K was taken from [20].

Details on Sections 4.2, 4.3 and 4.4. In these sections, the pretrained ResNet models are taken from the same sources as those in Section 4.1.

For pretraining PointNet++ on ImageNet1K, we utilize the PointNet++ SSG version [17]. We break the image into pixels and regard the group of pixels as a point-cloud with coordinates of x, y positions in the original image and appending $z = 1$ to all pixels. Then, we set center sampling number to 1024 and 256 for first and second stage, and the radius is set into 8 and 64, respectively. For each center point, we query 64 neighbouring points. The training recipe is also provided by PyTorch.

For ViT models, we directly take the pretrained weights from [4]. To apply it on ModelNet 3D Warehouse, we sample 256 centers and group 64 nearby points, regarding these as "point-cloud patches". Then, we use a linear embedding to project the point-cloud patches into a sequence, and ViT processes them the same as image patches. Except for the linear embedding and the final output classifier, all the models are kept the same as the original version. For the experiments on S3DIS and SemanticKITTI, the architectural detail of ResNet18 is shown in A.4 listing 1.2.

For SimpleView model, all the experiment settings are the same as [6]. The only difference is whether to use the pretrained ResNet18. For HRNetV2-W48, we directly use the ImageNet1K and Cityscape pretrained models from [21].

We conduct three trials on the few-shot experiments. For each trial, we change the random seed but keep all the other settings the same. To plot the training

[4] https://pytorch.org/vision/stable/models.html
[5] https://github.com/Alibaba-MIIL/ImageNet21K
[6] https://github.com/hirokatsukataoka16/FractalDB-Pretrained-ResNet-PyTorch
[7] https://github.com/HRNet/HRNet-Semantic-Segmentation/tree/pytorch-v1.1
[8] https://github.com/rgeirhos/Stylized-ImageNet
[9] https://github.com/wielandbrendel/bag-of-local-features-models
[10] https://github.com/mit-han-lab/torchsparse

Table 1. ResNet50 results (evaluated on ModelNet 3D Warehouse) of finetuning the mean and variance in batch normalization layers (BN) on different image-pretrained-models. IO (FIP-IO) indicates finetuning input and output layer. IOms indicates updating input and output layer, BN mean and variance. IOmsWb (FIP-IO+BN) indicates finetuning input, output layer, and the whole BN.

Layers	**Tiny-ImageNet**	**ImageNet1K**	**ImageNet21K**	**FractalDB1K**	**FractalDB10K**
IO	67.67	81.20	73.74	83.35	80.11
IOms	83.79	82.94	84.08	72.33	79.66
IOmsWb	89.99	89.87	90.80	89.26	89.34
From Scratch			90.32		

Table 2. ResNet18, 50, 152 results (evaluated on ModelNet 3D Warehouse) of finetuing the mean and variance in batch normalization layers.

Layers	**ResNet18**	**ResNet50**	**ResNet152**
IO	71.03	81.20	64.63
IOms	81.89	82.94	82.66
IOmsWb	88.75	89.87	90.44
From Scratch	90.39	90.32	90.28

speed curve, we directly use the training log without any other changes, such as smoothing.

B Additional Experiments

B.1 Finetuning the mean and variance of batch normalization.

For the first group of experiments, ResNet50 FIP either has IO or IO+BN finetuned. In addition to these two experimental settings, we also investigate finetuning input, output layers, and mean, variance of normalization layers, while fixing the convolution layer weights, normalization layer weights, and bias. The full experiment results with this extra setting are reported in Table 1 and 2. We can observe that compared with only finetuning input and output layers, updating mean and variance can also largely improve the performance of point-cloud recognition. As suggested in Section 2.4, we train batch normalizations to enhance the adaption between modalities.

B.2 Ablation study of inflating towards different directions.

We conduct experiments of inflating filters along different directions with the illustration figure shown in Figure 1 and the results shown in Table 3. We find that the performance is different when using different inflation methods. In particular, with ResNet50 pretrained on ImageNet1K, inflating along the x axis and the y axis leads to better performance compared with inflating along z axis for both FIP-IO and FIP-IO+BN. More importantly, the minimally finetuned FIP-IO+BN with inflating along the x and y axis even surpasses the training-from-scratch.

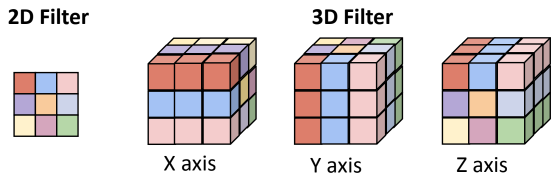

Fig. 1. Visualization of filter inflation along different axis.

Table 3. ModelNet 3D Warehouse results (Top-1 accuracy) of partially finetuning ResNet50 pretrained on ImageNet1K with inflation along the x, y, z axis.

Method	**x axis**	**y axis**	**z axis**
FIP-IO	82.17	81.73	81.20
FIP-IO+BN	90.44	90.84	89.87
From Scratch		90.32	

Table 4. ModelNet 3D Warehouse results (Top-1 accuracy) of finetuning ResNet50 pretrained on ImageNet1K with only subset of stages loaded.

Loaded Stages	**1**	**1 2**	**1 2 3**	**1 2 3 4**	**2 3 4**	**3 4**	**4**
FIP-ALL	89.91	90.36	90.64	90.92	91.09	90.19	89.99
From Scratch				90.32			

Table 5. Stability analysis of semi-supervised distillation experiments on top of ResNet34 on the ModelNet 3D Warehouse dataset.

Few-shot	**From scratch**	**PointInfoNCE**	**Hardest Contrastive**	**ImageNet1K pretrain (Ours)**
10-shot	72.7±1.2	74.3±1.3 (+**1.6**)	73.9±1.1 (+**1.2**)	74.6±1.1 (+**1.9**)
5-shot	62.2±0.7	64.5±1.4 (+**2.3**)	65.0±2.1 (+**2.8**)	65.4±1.4 (+**3.2**)
1-shot	30.8±2.4	36.5±1.8 (+**5.7**)	35.9±1.0 (+**5.1**)	38.3±2.0 (+**7.5**)

B.3 Ablation study of loading different stages of the image-pretrained model.

We investigate the effect of loading different subsets of stages. The results are shown in Table 4. In detail, we load the pretrained weights partially while keeping the other weights randomly initialized. We observe that excluding the weights of the first stage achieves the best performance, bringing 0.77 points improvement.

B.4 Stability analysis of the semi-supervised experiment.

For the semi-supervised experiment, we change the random seed and calculated the mean and standard deviation of three trials for each setting as shown in Table 5.

Table 6. Comparison with PointCLIP on 16-shot classification.

Method	1 view(Acc.%)	4 view(Acc.%)
PointCLIP (ResNet101)	75.53	82.17
FIP-ALL (ResNet101)	80.47	82.46

B.5 Comparison with other knowledge transfer methods.

We compared our method with PointCLIP[23], the most recent SOTA knowledge transfer approach in point-cloud domain. We show that with simply inflating ResNet101 pretrained on ImageNet1K, the 16-shot classification of Image2Point on ModelNet40 is better than PointCLIP, as shown in Table 6. We mimic PointCLIP to rotate point-cloud into different angles, and ensemble their results. Note that the FLOPs of Image2Point are less than PointCLIP. The results suggest that without a variety of tricks, our proposed method with simple inflation can achieve competitive and even better performance than knowledge-transfer methods.

B.6 Neural Collapse in the Embedding Layer.

[16] characterized neural collapse as training dynamics of overparameterized neural networks in which the feature embeddings of samples from the same class tend to concentrate around their class means. In this section, we briefly define neural collapse and evaluate it in our current setting. We refer the reader for additional details in [5,16].

Suppose we have a classification problem, in which we are provided with a training dataset $S = \cup_{c=1}^{C} S_c = \cup_{c=1}^{C} \{(x_{ci}, c)\}_{i=1}^{m_0}$ split into classes. We would like to train a neural network $h = e \circ q$, with $q : \mathbb{R}^n \to \mathbb{R}^p$ and $e : \mathbb{R}^p \to \mathbb{R}^C$ is a linear layer. The neural network is trained by minimizing cross-entropy loss between the one-hot encodings of its labels of samples in S and its logits. For additional details, see Section C.1.

Several definitions of neural collapse have been proposed in the literature. In this paper we work with a relatively simple definition that has been proposed in [5]. We start by defining the *class-distance normalized variance* (CDNV), which is a measure of clusterability of the feature embeddings. For a given feature map $q : \mathbb{R}^n \to \mathbb{R}^p$ and two distributions Q_1, Q_2 (of samples from two different classes) over $\mathcal{X}$, the [11] is defined in the following manner

$$V_q(Q_1, Q_2) = \frac{\mathrm{Var}_q(Q_1) + \mathrm{Var}_q(Q_2)}{2\|\mu_q(Q_1) - \mu_q(Q_2)\|^2}. \tag{1}$$

Essentially, this quantity measures to what extent the deviations of the embeddings $q(x)$ of samples coming from Q_1 and Q_2 are smaller than the distance

[11] The CDNV can be extended to finite sets $S_1, S_2 \subset \mathcal{X}$ by defining $V_q(S_1, S_2) = V_q(U[S_1], U[S_2])$.

between their means. Intuitively, if the deviations are very small in comparison with the distances, then we expect the embeddings to be clustered with respect to their class labels. Note, that this quantity is also scale-invariant, i.e., if we multiply q by $\alpha \neq 0$, then, the CDNV would not change for any pair Q_1, Q_2.

According to the definition in [5], *neural collapse* is defined in the following manner

$$\lim_{t \to \infty} \mathrm{Avg}_{i \neq j \in [l]} [V_{q_t}(S_i, S_j)] = 0, \tag{2}$$

where q_t is the embedding function after t epochs of training $e \circ q$. Intuitively, during train time, the feature embeddings of samples of the same class tend to concentrate around their class-means in comparison with their distance from the other classes.

Evaluating neural collapse. In this experiment we measure the clusterability of the feature embeddings of the penultimate layer of the pretrained model into classes, on both the source/pretraining task (i.e., ImageNet1K) and the target task (e.g., ModelNet40). In this experiment, we train a classifier of the form $\tilde{h} = \tilde{e} \circ q = \tilde{e} \circ f \circ \tilde{g}$ on ImageNet1K. We used ResNet50 as the architecture of h, where $\tilde{g}$ is the first convolutional layer of the model and $\tilde{e}$ is the top linear layer of the model. As a second step, we replace $\tilde{e}$ and $\tilde{g}$ with neural networks e (a linear layer) and g (a linear layer) and train them, along with retraining the batch normalization parameters of f on ModelNet40, resulting in a function $e \circ f' \circ g$. We denote by $\tilde{S}^{tr} = \cup_{c=1}^{k} \tilde{S}_c^{tr}$ the source training dataset and by $S^{tr} = \cup_{c=1}^{l} S_c^{tr}$ the target training dataset. Here, $\tilde{S}_c^{tr}$ and S_c^{tr} are the samples associated with the cth class. We also denote by $\tilde{S}^{val} = \cup_{c=1}^{k} \tilde{S}_c^{val}$ and $S^{val} = \cup_{c=1}^{k} S_c^{val}$ the corresponding validation datasets.

To measure the degree of clusterability of the feature embeddings, we consider multiple applications of the averaged CDNV. For measuring the clusterability of the features on the source task, we consider the CDNV of $f \circ \tilde{g}$ on the source train and validation datasets: $\mathrm{Avg}_{i \neq j \in [k]}[V_{f \circ \tilde{g}}(\tilde{S}_i^{tr}, \tilde{S}_j^{tr})]$ and $\mathrm{Avg}_{i \neq j \in [k]}[V_{f \circ \tilde{g}}(\tilde{S}_i^{val}, \tilde{S}_j^{val})]$. These results are reported in Table 7. Similarly, we also measure the CDNV of $f' \circ g$ on the source train and validation datasets: $\mathrm{Avg}_{i \neq j \in [k]}[V_{f' \circ g}(\tilde{S}_i^{tr}, \tilde{S}_j^{tr})]$ and $\mathrm{Avg}_{i \neq j \in [k]}[V_{f' \circ g}(\tilde{S}_i^{val}, \tilde{S}_j^{val})]$. In the main text, we report the CDNV on the validation set for $e \circ f' \circ g$ which reflects the property of the embedding's clusterability in a low-dimensional space.

As can be seen in Table 7, across all of the experiments, the values of the CDNV are lower than 1, meaning that the standard deviations of the embeddings per class are smaller in comparison with the distances between class means. Therefore, we encounter a scenario where the embeddings of samples are fairly separated into classes. In addition, we observe that the degree of collapse generalizes well to new samples, as the CDNV on the train and validation data are relatively similar.

B.7 Details of used architectures.

Table 7. CDNV of the pretrained models on ImageNet1K Training/validation set, and CDNV of training from scratch and FIP-IO+BN on ModelNet 3D Warehouse training/validation set

Models	ImageNet1K	From scratch	FIP-IO+BN ImageNet1K	FIP-IO+BN ImageNet21K
Training CDNV	0.63	0.37	0.71	0.47
Validation CDNV	0.66	0.43	0.68	0.60

```
Class 3DRes_cls(nn.Module):
    def __init__(self, res_block):
        # res_block means the residual block as same as the conventional ResNet.
        super().__init__()

        self.input_layer = nn.Sequential(
            sparse_conv3d(input_dim, layer1_Idim, k=3, s=1),
            sparse_bn(layer1_Idim))

        self.layer1 = inflated_resnet_layer1(
            res_block, layer1_Idim, layer1_Odim)
        self.layer2 = inflated_resnet_layer2(
            res_block, layer2_Idim, layer2_Odim)
        self.layer3 = inflated_resnet_layer3(
            res_block, layer3_Idim, layer3_Odim)
        self.layer4 = inflated_resnet_layer4(
            res_block, layer4_Idim, layer4_Odim)

        self.output_layer = nn.Sequential(
            global_average_pooling,
            nn.Linear(layer4_Odim, class_num),
            nn.bn(class_num))

    def forward(self, x):
        x = self.input_layer(x)
        x = self.layer1(x)
        x = self.layer2(x)
        x = self.layer3(x)
        x = self.layer4(x)
        return self.output_layer(x)
```

Listing 1.1. Pseudo code of inflated ResNet with linear input and output layers for classification.

```
Class 3DRes_seg(nn.Module):
    def __init__(self, res_block):
        # res_block means the residual block as same as the conventional ResNet.
        super().__init__()

        self.input_layer = nn.Sequential(
            sparse_conv3d(
                input_dim, layer1_Idim, k=3, s=1),
            sparse_bn(layer1_Idim),
            sparse_ReLU(True),
            sparse_conv3d(
                layer1_Idim, layer1_Idim, k=3, s=1),
            sparse_bn(layer1_Idim),
            sparse_ReLU(True),
            sparse_conv3d(
                layer1_Idim, layer1_Idim, k=3, s=2),
            sparse_bn(layer1_Idim),
            sparse_ReLU(True))

        self.layer1 = inflated_resnet_layer1(
            res_block, layer1_Idim, layer1_Odim)
        self.layer2 = inflated_resnet_layer2(
            res_block, layer2_Idim, layer2_Odim)
        self.layer3 = inflated_resnet_layer3(
            res_block, layer3_Idim, layer3_Odim)
        self.layer4 = inflated_resnet_layer4(
            res_block, layer4_Idim, layer4_Odim)

        self.up1 = sparse_deconv(
            layer4_Odim, layer4_Odim, k=2, s=2)
        self.decode1 = self.Sequential(
            res_block(layer4_Odim+layer3_Odim, layer3_Odim),
            res_block(layer3_Odim, layer3_Odim))

        self.up2 = sparse_deconv(
            layer3_Odim, layer3_Odim, k=2, s=2)
        self.decode2 = self.Sequential(
            res_block(layer3_Odim+layer2_Odim, layer2_Odim),
            res_block(layer2_Odim, layer2_Odim))

        self.up3 = sparse_deconv(
            layer2_Odim, layer2_Odim, k=2, s=2)
        self.decode3 = self.Sequential(
            res_block(layer2_Odim+layer1_Odim, layer1_Odim),
            res_block(layer1_Odim, layer1_Odim))

        self.up4 = sparse_deconv(
            layer1_Odim, layer1_Odim, k=2, s=2)
        self.decode4 = self.Sequential(
            res_block(layer1_Odim+layer1_Odim, layer1_Odim),
            res_block(layer1_Odim, layer1_Odim))

        self.output_layer = nn.Sequential(
            nn.Linear(layer1_Odim, class_num))

```

```
    def forward(self, x):
        x_i = self.input_layer(x)
        x1 = self.layer1(x_i)
        x2 = self.layer2(x1)
        x3 = self.layer3(x2)
        x4 = self.layer4(x3)

        x3_ = self.decoder1(cat(x3, self.up1(x4)))
        x2_ = self.decoder2(cat(x2, self.up2(x3_)))
        x1_ = self.decoder3(cat(x1, self.up3(x2_)))
        xi_ = self.decoder4(cat(x_i, self.up4(x1_)))
        return self.output_layer(xi_)
```

Listing 1.2. Pseudo code of inflated ResNet for segmentation.

C Theoretical Analysis

To motivate our approach, in this section we provide some theoretical support for the transfer of image domain to point-cloud domain described in the main text. Instead of specifying image and point-cloud in the analysis, we begin by introducing a formal framework for analyzing transfer learning between different modalities and classes. Note that the modalities should have grounded "relationships", or as illustrated below, have meaningful task similarity. For example, image and point-cloud are both visual representations of the real world that share mutual characteristics, such as content and shape that could be captured using neural networks encoders.

We begin by introducing a formal framework for analyzing transfer learning between different modalities and classes. Then, we analyze a simple toy example, in which it is possible to perfectly translate one modality to the other using a linear mapping. Finally, we consider a more realistic case, in which we assume that the two domains share a 'mutual semantic space' that encodes the content within samples from the two domains.

C.1 Problem Setup

We extend the transfer learning setting in [5]. We consider the problem of training a generic feature representation on a source classification task and transferring it to a target task. The two classification problems correspond to different modalities (*i.e.*, two different kinds of data representations) and consist of different sets of classes.

To model this problem, we assume that the target task is a k-class classification problem and the source task where the feature representation is learned on an l-class classification problem. Formally, the target task $T = (P, y)$ is defined by a distribution P over samples $x \in \mathcal{X}$, where $\mathcal{X} \subset \mathbb{R}^{d_1}$ is the instance space, along with a function $y : \mathcal{X} \to \mathcal{Y}_k$, where $\mathcal{Y}_k$ is a label space with cardinality k. To simplify the presentation, we use one-hot encoding for the label space, that is, the labels are represented by the unit vectors in $\mathbb{R}^k$, and $\mathcal{Y}_k = \{e_c : c = 1, \dots, k\}$ where $e_c \in \mathbb{R}^k$ is the cth standard unit vector in $\mathbb{R}^k$; with a slight abuse of notation, sometimes we will also write $y(x) = c$ instead of $y(x) = e_c$. For a sample x with distribution P, we denote by $P_c(\cdot) = \mathbb{P}[x \in \cdot \mid y(x) = c]$ the class distribution of x given $y(x) = c$. We consider balanced classes, *i.e.*, $\mathbb{P}[y(x) = c] = 1/k$.

The target task. A classifier $h \in \mathcal{H}$ is a mapping $h : \mathcal{X} \to \mathbb{R}_k$ that assigns a *soft* label to an input point $x \in \mathcal{X}$, and its performance on the target task is measured by the risk

$$L_P(h, y) = \mathbb{E}_{x \sim P}[\ell(h(x), y(x))], \tag{3}$$

where $\ell : \mathbb{R}^k \times \mathcal{Y}_k \to [0, \infty)$ is a loss function, defined as follows $\ell(u, v) = \mathbb{I}[\arg\max(u) = v]$.

Our goal is to learn a classifier h from some training data $S = \cup_{c=1}^k S_c = \cup_{c=1}^k \{(x_{ci}, c)\}_{i=1}^n$ of n independent and identically distributed (i.i.d.) samples drawn from each class P_c of P. However, when encountering a complicated classification problem and n is small, this is likely to be a hard task. To facilitate finding a good solution, we aim to find a classifier of the form $h = e \circ f \circ g$, where $f : \mathbb{R}^{p_1} \to \mathbb{R}^{p_2}$ is a feature map from a family of functions $\mathcal{F} \subset \{f' : \mathbb{R}^{p_1} \to \mathbb{R}^{p_2}\}$, $e \in \mathcal{E} \subset \{e' : \mathbb{R}^{p_2} \to \mathbb{R}^k\}$ is an affine function and g is a function from a family $\mathcal{G}_1 \subset \{g' : \mathbb{R}^{d_1} \to \mathbb{R}^{p_1}\}$. Namely, the feature map f is trained on a source problem, potentially of a different modality, where much more data is available, and then g and e are trained on S while freezing f. Intuitively, g and e are relatively 'simple' functions (*e.g.*, linear layers) that are task-specific. Concretely, e (the classifier) is responsible for translating f into a classifier between the classes in P and g (the adaptor) adapts f to the specific modality of P.

The source task. We assume that the source task helping to find f is an l-class classification problem over a different sample space $\mathcal{X}' \subset \mathbb{R}^{d_2}$. For example, $\mathcal{X}$ could be a set of 3D point-clouds, while $\mathcal{X}'$ is a set of 2D natural images. The source task $B = (\tilde{P}, \tilde{y})$ is defined by a distribution $\tilde{P}$ and function $\tilde{y} : \mathcal{X}' \to \mathcal{Y}_l$, and here we are interested in finding a classifier $\tilde{h} : \mathcal{X}' \to \mathbb{R}^l$ of the form $\tilde{h} = \tilde{e} \circ f \circ \tilde{g}$, where $\tilde{e} \in \tilde{\mathcal{E}} \subset \{e' : \mathbb{R}^p \to \mathbb{R}^l\}$ is an affine function over the feature space $f(\tilde{g}(\mathcal{X})) = \{f(\tilde{g}(x)) : x \in \mathcal{X}\}$ and $\tilde{g} \in \mathcal{G}_2 \subset \{g' : \mathbb{R}^{d_2} \to \mathbb{R}^{p_1}\}$ is an adaptor. Given a training dataset $\tilde{S} = \cup_{c=1}^l \tilde{S}_c = \cup_{c=1}^l \{(\tilde{x}_{ci}, c)\}_{i=1}^m$, all components of the classifier, denoted by $\tilde{g}$, f and $\tilde{e}$ are trained on $\tilde{S}$, with the goal of minimizing the cross-entropy loss in the source task. Following [5] we assume that $\tilde{S}_c$ are drawn i.i.d. from $\tilde{P}_c$ and that $\mathbb{P}[\tilde{y}(x) = c] = 1/l$.

Tasks similarity. In general, transferring between the source and target tasks is meaningless if the two tasks are extremely unrelated to each other. For instance, we should not expect to have any guarantee to transfer knowledge from very different tasks, such as voice separation and image segmentation.

Intuitively, we think of the classes of the target (source) task as being of *'similar character'*. To formalize this intuition, we simply assume that the target (source) classes are i.i.d. samples of a distribution $\mathcal{D}_1$ over $\mathcal{C}_1$ ($\mathcal{D}_2$ over $\mathcal{C}_2$).

In [5] they assumed that the source and target classes differ, but share the same underlying distribution, *i.e.*, $\mathcal{D}_1 = \mathcal{D}_2$. Since in this work we intend to study transfer between different modalities, $\mathcal{D}_1$ need not be the same as $\mathcal{D}_2$. To formally define notions of similarity between domains, we first assume the existence of an invertible mapping $F : \mathcal{C}_1 \to \mathcal{C}_2$, such that, $\hat{P}_c := F(P_c) \sim \mathcal{D}_2$ for $P_c \sim \mathcal{D}_1$. In a sense, $\mathcal{D}_1$ and $\mathcal{D}_2$ share the same set of categories, encoded with different kinds of modalities. The mapping F takes a certain class $P_c \in \mathcal{C}_1$ and maps it to its analogous class $\hat{P}_c \in \mathcal{C}_2$ in the second domain. For a given target task $T = (P, y)$ with distribution P, classes $\{P_c\}_{c=1}^{k}$ and function $y : \mathcal{X} \to \mathbb{R}^k$, we denote $A = (\hat{P}, \hat{y})$ the corresponding analogous task within the second domain, where $\hat{P}_c = F(P_c)$ (not to be confused with the source task $B = (\tilde{P}, \tilde{y})$). In general, F could be an arbitrary mapping between the classes. Therefore, to concretely relate between $\mathcal{D}_1$ and $\mathcal{D}_2$ we will have to make additional assumptions about the relationship between P and $\hat{P}$. The specific relationship between P_c and $\hat{P}_c$ will be explicitly defined near the presentation of each result.

Evaluation process. As a next step, we would like to evaluate the performance of the pretrained feature map f on the target task. To do so, we evaluate its expected performance over the distribution of binary classification target tasks

$$L^k_{\mathcal{D}_1}(f) = \mathbb{E}_{P_1 \neq \ldots \neq P_k \sim \mathcal{D}_1} \mathbb{E}_{S_1,\ldots,S_k}[L_P(e_S \circ f \circ g_S, y)], \tag{4}$$

where $S_c \sim P_c^n$ and e_S, g_S are the outputs of a learning algorithm that trains $e \in \mathcal{E}$ and $g \in \mathcal{G}_1$ to fit $e \circ f \circ g$ to the dataset S while freezing f. For simplicity, in this work we focus on $k = 2$ and denote $L_{\mathcal{D}_1}(f) = L^2_{\mathcal{D}_1}(f)$, even though the analysis could be readily extended to $k > 2$. Note that several implementations of mappings $S \mapsto e_S, g_S$ are possible. For simplicity, in this work, we choose $\arg\max \circ e_S$ to be the *'nearest empirical mean classifier'* and g_S to be an empirical risk minimizer. Formally, for a given embedding function h, we consider the linear function $e^S_h(z) = (\langle z, 2\mu_h(S_c)\rangle - \|\mu_h(S_c)\|^2)_{c=1,2}$. In this case, $\arg\max \circ e^S_h(z) = \min_{c=1,2} \|z - \mu_h(S_c)\|$ forms a nearest empirical mean classification rule that classifies a vector z as 1 if it is closer to the empirical embeddings mean $\mu_h(S_1) = \mathrm{Avg}_{x\sim S_1}[h(x)]$ than to $\mu_h(S_2) = \mathrm{Avg}_{x\sim S_2}[h(x)]$. In addition, $g_S = \arg\min_{g\in\mathcal{G}} L_X(e^S_{f\circ g} \circ f \circ g, y)$, where $X = \cup_{c=1}^{k}\{x_{ci}\}_{i=1}^{n}$ and $e_S = e_{f\circ g_S}$ is the corresponding nearest empirical mean classifier. Notice that while the feature map f is evaluated on the distribution of target tasks determined by classes taken from $\mathcal{D}_1$, the training of f, as described above, is fully agnostic of this target.

To summarize, in the proposed setting we train a feature map f as part of a classification model $\tilde{e} \circ f$ to fit some source data corresponding to a set of source classes $\tilde{P}_1, \ldots, \tilde{P}_l$ using dataset $\tilde{S} = \cup_{c=1}^{l}\tilde{S}_c$. At the second stage, f's performance is evaluated against a randomly selected set of target classes $P_1, \ldots, P_k$ the differ by *content* (i.e., different categories) and *modality* (e.g., 2D and 3D point clouds) by training an adaptor g and a linear classifier e based on the available data $S = \cup_{c=1}^{k} S_c$. Therefore, in this work we deal with *two modes of transfer*. First, we would like to train f to be a generic feature map that can be used to distinguish between many different categories. The second deals with the ability to adapt f

from one modality to another using minimal efforts. Intuitively, if the pretrained feature map f enjoys both qualities, we expect $L_{\mathcal{D}_1}(f)$ to be small. This is what we show in Theorem 1.

Notation. Throughout the analysis, we use the following notations. For an integer $k \geq 1$, $[k] = \{1, \dots, k\}$. For any real vector z, $\|z\|$ denotes its Euclidean norm. For a given set $A = \{a_1, \dots, a_n\} \subset B$ and a function $u \in \mathcal{U} \subset \{u' : B \to \mathbb{R}\}$, we define $u(A) = \{u(a_1), \dots, u(a_n)\}$ and $\mathcal{U}(A) = \{u(A) : u \in \mathcal{U}\}$. Let Q be a distribution over $\mathcal{X} \subset \mathbb{R}^d$ and $u : \mathcal{X} \to \mathbb{R}^p$. We denote by $\mu_u(Q) = \mathbb{E}_{x \sim Q}[u(x)]$ and by $\mathrm{Var}_u(Q) = \mathbb{E}_{x \sim Q}[\|u(x) - \mu_u(Q)\|^2]$ the mean and variance of $u(x)$ for $x \sim Q$. For A above, we denote by $\mathrm{Avg}_{i=1}^n[a_i] = \mathrm{Avg}\, A = \frac{1}{n}\sum_{i=1}^n a_i$ the average of A. For a finite set A, we denote by $U[A]$ the uniform distribution over A. We denote by $\mathbb{I} : \{\mathrm{True}, \mathrm{False}\} \to \{0, 1\}$ the indicator function. For a given distribution P over $\mathcal{X}$ and a measurable function $f : \mathcal{X} \to \mathcal{X}'$, we denote the distribution of $f(x)$ by $f \circ P$. For two classes of functions $\mathcal{G} = \{g' : \mathbb{R}^{d_1} \to \mathbb{R}^{d_2}\}$ and $\mathcal{F} = \{f' : \mathbb{R}^{d_2} \to \mathbb{R}^{d_3}\}$, we denote $\mathcal{F} \circ \mathcal{G} = \{f \circ g \mid f \in \mathcal{F}, g \in \mathcal{G}\}$.

D Theoretical Results

D.1 Case 1

As a toy example, we first consider the case where the first domain (target) can be translated into the second domain (source), using some simple function g^*. For this purpose, we assume that $\mathcal{G}_1$ is decomposed into $\mathcal{G}_1 = \mathcal{G}_1'' \circ \mathcal{G}_1'$, where $\mathcal{G}_1' \subset \{g' : \mathbb{R}^{d_1} \to \mathbb{R}^{d_2}\}$ and $\mathcal{G}_1'' \subset \{g' : \mathbb{R}^{d_2} \to \mathbb{R}^{p_1}\}$. Intuitively, the functions $g \in \mathcal{G}_1$ are decomposed into sub-architectures $g_1' : \mathbb{R}^{d_1} \to \mathbb{R}^{d_2}$ and $g_1'' : \mathbb{R}^{d_2} \to \mathbb{R}^{p_1}$. In addition, we assume there exists a function $g^* \in \mathcal{G}_1'$, such that $g^* \circ P \sim \mathcal{D}_2$ for $P \sim \mathcal{D}_1$ (where $g \circ P$ is the distribution of $g(x)$ for $x \sim P$). In addition, we assume that $\mathcal{G}_2 \subset \mathcal{G}_1''$. Hence, for each candidate function $\tilde{g} \in \mathcal{G}_2$ that could have been selected when training the classifier $h = \tilde{e} \circ f \circ \tilde{g}$, there exists a function $g = \tilde{g} \circ g^* \in \mathcal{G}_1$ that maps P_c into $g \circ P_c = \tilde{g} \circ \tilde{P}_c$. Intuitively, any representation $\tilde{g} \circ \tilde{P}$ of the distribution $\tilde{P}$ could be implemented as $g \circ P$ for some $g \in \mathcal{G}_1$. While g^* is unknown to the learning algorithm, as we will see in Theorem 1, this simplifying assumption makes it possible to easily transfer between the two domains.

The following theorem provides an upper bound on the expected error of a pretrained feature map f in terms of the expected CDNV between pairs of classes $\tilde{P}_i$ and $\tilde{P}_j$ from $\mathcal{D}_2$ and the empirical Rademacher complexity of the class $\mathcal{H}_f = \{\arg\max \circ e \circ f \circ g \mid (e, g) \in \mathcal{E} \times \mathcal{G}_1\}$. The Rademacher complexity is a measure of generalization that quantifies the ability of a class of functions to fit noise. Formally, for a given set $X = \{x_i\}_{i=1}^n \subset \mathbb{R}^d$ and set of functions $\mathcal{H} \subset \{h' \mid h' : \mathbb{R}^d \to \mathbb{R}\}$, the empirical Rademacher complexity (see [14]) of $\mathcal{H}$ is defined as follows

$$\mathrm{Rad}_X(\mathcal{H}) = \mathbb{E}_\sigma \left[\sup_{h \in \mathcal{H}} \frac{1}{n} \sum_{i=1}^n \sigma_i \cdot f(x_i) \right], \qquad (5)$$

where $\sigma = (\sigma_1, \ldots, \sigma_n)$ are i.i.d. uniformly distributed over $\{\pm 1\}$. The empirical Rademacher complexity can lead to tighter bounds than those based on other measures of complexity such as the VC-dimension [8]. It also has the added advantage that it is data-dependent and can be measured from finite samples.

Theorem 1. *In the setting above. For any tuple $(f, \tilde{g}) \in \mathcal{F} \times \mathcal{G}$, we have:*

$$\begin{aligned} L_{\mathcal{D}_1}(f) \;\leq\; & 16\mathbb{E}_{\tilde{P}_1 \neq \tilde{P}_2 \sim \mathcal{D}_2}\left[\frac{V_{f\circ\tilde{g}}(\tilde{P}_1, \tilde{P}_2)}{s(f\circ\tilde{g}, \tilde{P}_1, \tilde{P}_2)}\right] + 6\sqrt{\frac{\log(4n)}{2n}} + \frac{2}{n} \\ & + 2\mathbb{E}_{P_c \sim \mathcal{D}_1}\mathbb{E}_{X_c \sim P_c^n}\left[\mathrm{Rad}_{X_c}(\mathcal{H}_f)\right], \end{aligned}$$

where $s(f, P_1, P_2) = p_2$ if $f \circ P_1$ and $f \circ P_2$ are spherically symmetric and $s(f, P_1, P_2) = 1$ otherwise.

The theorem above provides an upper bound on the expected error $L_{\mathcal{D}_1}(f)$ of a pretrained feature map f against classification tasks generated using classes from $\mathcal{D}_1$. The bound holds uniformly for any pair $(f, \tilde{g}) \in \mathcal{F} \times \mathcal{G}$. Hence, we think of f and $\tilde{g}$ as the pretrained functions that were obtained by training the classifier $\tilde{e} \circ f \circ \tilde{g}$ to fit the source data $\tilde{S} = \cup_{c=1}^{l} \tilde{S}_c$. The bound is decomposed into several parts. The first term is (approximately) proportional to the CDNV between pairs of randomly selected classes from $\mathcal{D}_2$. Namely, it measures the extent of neural collapse we encounter between randomly selected pairs of source classes $\tilde{P}_1 \neq \tilde{P}_2 \sim \mathcal{D}_2$.

As mentioned in Section B.6 in the regime of neural collapse, we expect $\mathrm{Avg}_{i\neq j\in[l]}\left[V_{f\circ\tilde{g}}(\tilde{S}_i, \tilde{S}_j)\right]$ to be small. In addition, by Propositions 1 in [5], under certain circumstances if the number of samples per source class m is large enough, we also expect $\mathrm{Avg}_{i\neq j\in[l]}\left[V_{f\circ\tilde{g}}(\tilde{P}_i, \tilde{P}_j)\right]$ to be small. Finally, by Propositions 2 in [5], if the number of source classes l is also large, we should also expect $\mathbb{E}_{\tilde{P}_i \neq \tilde{P}_j \sim \mathcal{D}_2}[V_{f\circ\tilde{g}}(\tilde{P}_i, \tilde{P}_j)]$ to be small. As a side note, a smaller bound is obtained when $f \circ \tilde{g}$ projects the classes $\tilde{P}_c \sim \mathcal{D}_2$ to spherically symmetric distributions as $s(f \circ \tilde{g}, \tilde{P}_1, \tilde{P}_2) = p_2$ if $f \circ \tilde{g} \circ \tilde{P}_c$ are spherically symmetric. For estimations of $\mathrm{Avg}_{i\neq j\in[l]}\left[V_{f\circ\tilde{g}}(\tilde{S}_i, \tilde{S}_j)\right]$ and $\mathrm{Avg}_{i\neq j\in[l]}\left[V_{f\circ\tilde{g}}(\tilde{P}_i, \tilde{P}_j)\right]$, see Section B.6, in which $\tilde{S}_c$ is denoted by $\tilde{S}_c^{tr}$ and the class distributions $\tilde{P}_c$ are approximated with validation sets $\tilde{S}_c^{val}$.

The second group of terms includes the (expected) Rademacher complexity of the class $\mathcal{H}_f$ and additional terms that scale as $\mathcal{O}\left(\sqrt{\log(n)/n}\right)$. The Rademacher complexity $\mathrm{Rad}_S(\mathcal{H})$ of a class $\mathcal{H}$ of neural network classifiers $h : \mathbb{R}^d \to \{0, 1\}$ typically scales as $\mathcal{O}\left(\sqrt{N/n}\right)$, where N polynomially depends on the number of trainable parameters and n is the number of samples. Therefore, in standard settings, we expect $\mathrm{Rad}_S(\mathcal{H}_f)$ to scale as $\mathcal{O}\left(\sqrt{N/n}\right)$, where N polynomially depends on the number of parameters in e and g. On the other hand, a standard Rademacher complexity generalization bound would yield a dependence on the number of parameters existing in the full network $e \circ f \circ g$ (including the ones in f). Since typically f contains most of the trainable parameters in the neural network, this allows us to significantly reduce the sample complexity of the target task.

Proof. To prove this theorem, we fix a target task $T = (P, y)$ with a pair of classes P_1, P_2 and a pretrained feature map $f \circ \tilde{g}$. By (cf. [14], Theorem 3.5), for any $c = 1, 2$, with probability at least $1 - \frac{1}{4n}$ over the selection of S_c, for any pair $(e, g) \in \mathcal{E} \times \mathcal{G}_1$, we have

$$|L_{P_c}(e \circ f \circ g, y) - L_{X_c}(e \circ f \circ g, y)| \leq 2\,\mathrm{Rad}_{X_c}(\mathcal{H}_f) + 3\sqrt{\frac{\log(4n)}{2n}}, \tag{6}$$

where X_c consists of the samples in S_c excluding their labels and $X = X_1 \cup X_2$. By union bound over $c = 1, 2$, with probability at least $1 - \frac{1}{2n}$ over the selection of S, the following inequality holds uniformly for all $(e, g) \in \mathcal{E} \times \mathcal{G}_1$,

$$|L_P(e \circ f \circ g, y) - L_X(e \circ f \circ g, y)| \ \leq \ \sum_{c=1,2} \mathrm{Rad}_{X_c}(\mathcal{H}_f) + 3\sqrt{\frac{\log(4n)}{2n}}. \tag{7}$$

Hence, with probability at least $1 - \frac{1}{2n}$ over the selection of S,

$$\begin{aligned} L_P(e_S \circ f \circ g_S, y) \ &\leq \ L_X(e_S \circ f \circ g_S, y) \\ &\quad + \sum_{c=1,2} \mathrm{Rad}_{X_c}(\mathcal{H}_f) + 3\sqrt{\frac{\log(4n)}{2n}}. \end{aligned} \tag{8}$$

Let $\mathcal{E}' = \left\{e(z) = (-\|z - \mu_c\|^2)_{c=1,2} = (\langle z, 2\mu_c\rangle - \|\mu_c\|^2)_{c=1,2} \mid \mu_1, \mu_2 \in \mathbb{R}^{p_2}\right\} \subset \mathcal{E}$. In addition, we let $e_P(z) = (-\|z - \mu_{f\circ\tilde{g}\circ g^*}(P_c)\|^2)_{c=1,2}$. Since $\tilde{g} \circ g^* \in \mathcal{G}_2 \circ \mathcal{G}_1' \in \mathcal{G}_1'' \circ \mathcal{G}_1' = \mathcal{G}_1$, we have

$$\begin{aligned} L_X(e_S \circ f \circ g_S, y) \ &= \ \inf_{g\in\mathcal{G}_1} \inf_{e\in\mathcal{E}'} L_X(e \circ f \circ g, y) \\ &\leq \ \inf_{e\in\mathcal{E}'} L_X(e \circ f \circ \tilde{g} \circ g^*, y) \\ &\leq \ L_X(e_P \circ f \circ \tilde{g} \circ g^*, y). \end{aligned} \tag{9}$$

In particular, since the loss function is bounded in $[0, 1]$ and the above event holds with probability at least $1 - \frac{1}{2n}$, by taking expectation over S, we obtain the following inequality

$$\begin{aligned} &\mathbb{E}_S\left[L_P(e_S \circ f \circ g_S, y)\right] \\ &\quad \leq \ \mathbb{E}_S\left[L_S(e_P \circ f \circ \tilde{g} \circ g^*, y)\right] + 3\sqrt{\frac{\log(4n)}{2n}} + \frac{1}{2n} + \sum_{c=1,2} \mathrm{Rad}_{X_c}(\mathcal{H}_f) \\ &\quad = \ \mathbb{E}_S\left[L_S(e_P \circ f \circ \tilde{g} \circ g^*, y)\right] + 3\sqrt{\frac{\log(4n)}{2n}} + \frac{1}{2n} + \sum_{c=1,2} \mathrm{Rad}_{X_c}(\mathcal{H}_f). \end{aligned} \tag{10}$$

Finally, we can take expectation over the selection of P_1, P_2 on both sides of the inequality to obtain that

$$\begin{aligned} L_{\mathcal{D}_1}(f) &= \mathbb{E}_{P_1 \neq P_2 \sim \mathcal{D}_1} \mathbb{E}_S \left[L_P(e_S \circ f \circ g_S, y) \right] \\ &\leq \mathbb{E}_{P_1 \neq P_2} \left[L_P(e_P \circ f \circ \tilde{g} \circ g^*, y) \right] \\ &\quad + 2\mathbb{E}_{P_1 \neq P_2 \sim \mathcal{D}_1} \mathbb{E}_{X_c} \left[\mathrm{Rad}_{X_c}(\mathcal{H}_f) \right] + 3\sqrt{\frac{\log(4n)}{2n}} + \frac{1}{2n}. \end{aligned} \tag{11}$$

Finally, since for any given distribution P, we have $g^* \circ P \sim \mathcal{D}_1$ for $P \sim \mathcal{D}_2$, by Proposition 5 in [5], we obtain that

$$\begin{aligned} \mathbb{E}_{P_1 \neq P_2} \left[L_P(e_P \circ f \circ \tilde{g} \circ g^*, y) \right] &= \mathbb{E}_{\hat{P}_1 \neq \hat{P}_2} \left[L_{\hat{P}}(e_P \circ f \circ \tilde{g}, \hat{y}) \right] \\ &\leq \frac{16 V_{f \circ \tilde{g}}(\hat{P}_1, \hat{P}_2)}{s(f \circ \tilde{g}, \hat{P}_1, \hat{P}_2)}. \end{aligned} \tag{12}$$

D.2 Case 2

In the previous section, we assumed existence of a mapping $g^* \in \mathcal{G}'_1$, such that $g^* \circ P_c \sim \mathcal{D}_2$ for $P_c \sim \mathcal{D}_1$. However, this assumption is typically violated in practice [9,19,22]. Therefore, following the Unsupervised Domain Adaptation literature [1,12], we use a relaxed assumption that there is a 'shared representation space' for both domains. Informally, the two domains can be mapped to a shared space, in which classification into classes is possible. Variations of this assumption are algorithmically and theoretically used in multiple areas of computer vision [2,7,10,11,18,24].

Formally, we assume the existence of two mappings $g^* \in \mathcal{G}_1$ and $\tilde{g}^* \in \mathcal{G}_2$ that satisfy $g^* \circ P_c \approx \tilde{g}^* \circ \hat{P}_c$ in expectation over $P_c \sim \mathcal{D}_1$. To formalize this assumption, we make use of the notion of discrepancy [1,3]. Namely, for a given set $\mathcal{V}$ of functions $v : \mathcal{X} \to \mathbb{R}$, we define the discrepancy between two distributions P_1 and P_2 over $\mathcal{X}$, as $\mathrm{disc}_{\mathcal{V}}(P_1, P_2) = \sup_{h \in \mathcal{V}} |\mathbb{E}_{x \sim P_1}[h(x)] - \mathbb{E}_{x \sim P_2}[h(x)]|$. The discrepancy, or Integral Probability Metric (IPM) [15], is a pseudo-metric between distributions. Namely, we (implicitly) assume that $\mathbb{E}_{P_c \sim \mathcal{D}_1} \left[\mathrm{disc}_{\mathcal{V}}(g^* \circ P_c, \tilde{g}^* \circ \hat{P}_c) \right]$ is small, where $\mathcal{V}$ is some class of binary functions (to be defined in the proof).

To capture the intuition that one can classify the samples into classes from the representation space $g^*(\mathcal{X}) \approx \tilde{g}^*(\mathcal{X}')$, we assume that the following term

$$\mathbb{E}_{P_1 \neq P_2} \left[\inf_{(e,f) \in \mathcal{E} \times \mathcal{F}} L_P(e \circ f \circ g^*, y) + L_{\hat{P}}(e \circ f \circ \tilde{g}^*, \hat{y}) \right] \tag{13}$$

is small. In words, in expectation over the selection of P and $\hat{P}$, the correct labels $y(x)$ and $\hat{y}(x)$ can be simultaneously recovered using classifiers $e \circ f \circ g^*$ and $e \circ f \circ \tilde{g}^*$, for some $e \circ f \in \mathcal{E} \circ \mathcal{F}$.

Finally, as a technicality, we assume that the pretrained adaptor $\tilde{g}$ can be represented as $\tilde{g} = u \circ \tilde{g}^*$, for some function $u \in \mathcal{U}$, where $\mathcal{U} \subset \{u' : \mathbb{R}^{p_1} \to \mathbb{R}^{p_1}\}$, such that, $\mathcal{U} \circ \mathcal{G}_1 \subset \mathcal{G}_1$. For example, if $\mathcal{G}_1$ is a class of neural networks, ending with a linear layer and $\mathcal{U}$ is the set of linear mappings $u : \mathbb{R}^{d_1} \to \mathbb{R}^{d_1}$, then indeed we have $\mathcal{U} \circ \mathcal{G}_1 \subset \mathcal{G}_1$.

Theorem 2. *In the setting above. For any pair $(f, \tilde{g}) \in \mathcal{F} \times \mathcal{G}$, such that $\tilde{g} = u \circ \tilde{g}^*$, we have:*

$$\begin{aligned} L_{\mathcal{D}_1}(f) \leq 16\mathbb{E}_{\tilde{P}_1 \neq \tilde{P}_2 \sim \mathcal{D}_2}\left[\frac{V_{f\circ\tilde{g}}(\tilde{P}_i, \tilde{P}_j)}{s(f\circ\tilde{g}, \tilde{P}_1, \tilde{P}_2)}\right] + 2\mathbb{E}_{P_c}\mathbb{E}_{X_c \sim P_c^n}[\mathrm{Rad}_{X_c}(\mathcal{H}_f)] \\ + \mathbb{E}_{P_c}[\mathrm{disc}_{\mathcal{V}}(g^* \circ P_c, \tilde{g}^* \circ \hat{P}_c)] + 3\sqrt{\frac{\log(4n)}{2n}} + \frac{1}{2m} \\ + \mathbb{E}_{P_1 \neq P_2}\left[\inf_{(e,f)\in\mathcal{E}\times\mathcal{F}} L_P(e\circ f\circ g^*, y) + L_{\hat{P}}(e\circ f\circ\tilde{g}^*, \hat{y})\right], \end{aligned}$$

where $s(f, P_1, P_2) = p_2$ if $f \circ P_1$ and $f \circ P_2$ are spherically symmetric and $s(f, P_1, P_2) = 1$ otherwise.

The above theorem provides an upper bound on the expected error of the pretrained feature map f against binary classification target tasks $T = (P, y)$. In this theorem, we assume that f has been trained along with an adaptor $\tilde{g}$ that can be represented as $u \circ \tilde{g}^*$, where u is a linear mapping.

The upper bound is decomposed into multiple parts. Similar to the previous bound it sums the expected CDNV between pairs of classes $\tilde{P}_1, \tilde{P}_2$, the Rademacher complexity of $\mathcal{H}_f$ and additional terms scaling as $\mathcal{O}(\sqrt{\log(n)/n})$. As discussed in Section D.1, we expect these terms to be small. In addition, the bound also includes the expected discrepancy between $g^* \circ P_c$ and $\tilde{g}^* \circ \hat{P}_c$ that measures to what extent the two adaptors g^* and $\tilde{g}^*$ can map the distributions P_c and $\hat{P}_c$ to the same space. Finally, the last term measures to what extent we can actually recover the class label from representations taken from the shared space $g^* \circ P_c \approx \tilde{g}^* \circ \hat{P}_c$. As mentioned above, we explicitly assume that these terms are small. Intuitively, these terms are small when the two domains share a mutual semantic space $g^* \circ P_c \approx \tilde{g}^* \circ \hat{P}_c$ that encodes the content within samples from the two domains.

The proof of this theorem is based on the analysis of [5], the theory of Unsupervised Domain Adaptation [1,12,13] and Rademacher complexities [14].

Proof. Let $(\hat{e}, \hat{f}) \in \mathcal{E} \times \mathcal{F}$ be any pair of functions. Let u be a linear mapping, such that, $\tilde{g} = u \circ \tilde{g}^*$. To prove this theorem, we start by considering a pair of target classes P_1, P_2. Since the loss function is bounded in $[0, 1]$, for any $c = 1, 2$, by (cf. [14], Theorem 3.3) with probability at least $1 - \frac{1}{4m}$ over the selection of S_c, for any pair $(e, g) \in \mathcal{E} \times \mathcal{G}$, we have

$$L_{P_c}(e\circ f\circ g, y) \;\leq\; L_{X_c}(e\circ f\circ g, y) + 3\sqrt{\frac{\log(4m)}{2m}} + 2\,\mathrm{Rad}_{X_c}(\mathcal{H}_f). \tag{14}$$

By union bound over $c = 1, 2$, with probability at least $1 - \frac{1}{2m}$ over the selection of $S = S_1 \cup S_2$, for any pair $(e, g) \in \mathcal{E} \times \mathcal{G}$, we have

$$L_P(e \circ f \circ g, y) \;\leq\; L_S(e \circ f \circ g, y) + 3\sqrt{\frac{\log(4m)}{2m}} + \sum_{c=1,2} \mathrm{Rad}_{S_c}(\mathcal{H}_f). \tag{15}$$

Hence, with probability at least $1 - \frac{1}{2m}$ over the selection of S,

$$\begin{aligned} L_P(e_S \circ f \circ g_S, y) \;\leq\; & L_S(e_S \circ f \circ g_S, y) + 3\sqrt{\frac{\log(4m)}{2m}} \\ & + \sum_{c=1,2} \mathrm{Rad}_{S_c}(\mathcal{H}_f). \end{aligned} \tag{16}$$

Let $\mathcal{E}' = \left\{ e(z) = (-\|z - \mu_c\|^2)_{c=1,2} = (\langle z, 2\mu_c\rangle - \|\mu_c\|^2)_{c=1,2} \mid \mu_1, \mu_2 \in \mathbb{R}^{p_2} \right\} \subset \mathcal{E}$. In addition, we let $e_{\hat{P}}(z) = (-\|z - \mu_{f\circ\tilde{g}}(\hat{P}_c)\|^2)_{c=1,2}$. We notice that

$$\begin{aligned} L_S(e_S \circ f \circ g_S, y) \;&=\; \inf_{g\in\mathcal{G}_1} \inf_{e\in\mathcal{E}'} L_S(e \circ f \circ g, y) \\ &\leq\; \inf_{e\in\mathcal{E}'} L_S(e \circ f \circ u \circ g^*, y) \\ &\leq\; L_S(e_{\hat{P}} \circ f \circ u \circ g^*, y). \end{aligned} \tag{17}$$

Since the loss function is bounded in $[0, 1]$ and the above event holds with probability at least $1 - \frac{1}{2m}$, by taking expectation over S, we have the following

$$\begin{aligned} \mathbb{E}_S\left[L_P(e_S \circ f \circ g_S, y)\right] \leq L_P(e_{\hat{P}} \circ f \circ u \circ g^*, y) + 3\sqrt{\frac{\log(4m)}{2m}} + \frac{1}{2m} \\ + \sum_{c=1,2} \mathbb{E}_{S_c}\left[\mathrm{Rad}_{S_c}(\mathcal{H}_f)\right] \end{aligned} \tag{18}$$

In addition,

$$\begin{aligned} L_P(e_{\hat{P}} \circ f \circ u \circ g^*, y) \;\leq\; & L_P(e_{\hat{P}} \circ f \circ u \circ g^*, \hat{e} \circ \hat{f} \circ g^*) \\ & + L_P(\hat{e} \circ \hat{f} \circ g^*, y) \\ \leq\; & L_{\hat{P}}(e_{\hat{P}} \circ f \circ u \circ \tilde{g}^*, \hat{e} \circ \hat{f} \circ \tilde{g}^*) \\ & + L_P(\hat{e} \circ \hat{f} \circ g^*, y) \\ & + \mathrm{disc}_{\mathcal{V}}(g^* \circ P, \tilde{g}^* \circ \hat{P}) \\ \leq\; & L_{\hat{P}}(e_{\hat{P}} \circ f \circ u \circ \tilde{g}^*, \hat{y}) \\ & + L_{\hat{P}}(\hat{e} \circ \hat{f} \circ \tilde{g}^*, \hat{y}) + L_P(\hat{e} \circ \hat{f} \circ g^*, y) \\ & + \mathrm{disc}_{\mathcal{V}}(g^* \circ P, \tilde{g}^* \circ \hat{P}), \end{aligned} \tag{19}$$

where $\mathcal{V} = \{\mathbb{I}[e_1 \circ f_1 \circ u_1 \neq e_2 \circ f_2] \mid e_1, e_2 \in \mathcal{E}, f_1, f_2 \in \mathcal{F}, u_1 \in \mathcal{U}\}$. In particular, we can take infimum over $(\hat{e}, \hat{f}) \in \mathcal{E} \times \mathcal{F}$ to obtain

$$\begin{aligned} L_P(e_{\hat{P}} \circ f \circ u \circ g^*, y) \ &\leq \ L_{\hat{P}}(e_{\hat{P}} \circ f \circ u \circ \tilde{g}^*, \hat{y}) \\ &\quad + \inf_{e,f} \left[L_{\hat{P}}(e \circ f \circ \tilde{g}^*, \hat{y}) + L_P(e \circ f \circ g^*, y) \right] \\ &\quad + \mathrm{disc}_{\mathcal{V}}(g^* \circ P, \tilde{g}^* \circ \hat{P}) \\ &= \ L_{\hat{P}}(e_{\hat{P}} \circ f \circ \tilde{g}, \hat{y}) \\ &\quad + \inf_{e,f} \left[L_{\hat{P}}(e \circ f \circ \tilde{g}^*, \hat{y}) + L_P(e \circ f \circ g^*, y) \right] \\ &\quad + \mathrm{disc}_{\mathcal{V}}(g^* \circ P, \tilde{g}^* \circ \hat{P}). \end{aligned} \tag{20}$$

Next, we can take expectation over the selection of P on both sides of the inequality to obtain that

$$\begin{aligned} &\mathbb{E}_{P_1 \neq P_2} \left[L_P(e_{\hat{P}} \circ f \circ u \circ g^*, y) \right] \\ &\leq \ \mathbb{E}_{P_1 \neq P_2} \left[L_{\hat{P}}(e_{\hat{P}} \circ f \circ \tilde{g}, \hat{y}) \right] \\ &\quad + \mathbb{E}_{P_1 \neq P_2} \left[\inf_{e,f} \left[L_{\hat{P}}(e \circ f \circ \tilde{g}^*, \hat{y}) + L_P(e \circ f \circ g^*, y) \right] \right] \\ &\quad + \mathbb{E}_{P_1 \neq P_2} \left[\mathrm{disc}_{\mathcal{V}}(g^* \circ P, \tilde{g}^* \circ \hat{P}) \right] \\ &= \ \mathbb{E}_{\hat{P}_1 \neq \hat{P}_2} \left[L_{\hat{P}}(e_{\hat{P}} \circ f \circ \tilde{g}, \hat{y}) \right] \\ &\quad + \mathbb{E}_{P_1 \neq P_2} \left[\inf_{e,f} \left[L_{\hat{P}}(e \circ f \circ \tilde{g}^*, \hat{y}) + L_P(e \circ f \circ g^*, y) \right] \right] \\ &\quad + \mathbb{E}_{P_1 \neq P_2} \left[\mathrm{disc}_{\mathcal{V}}(g^* \circ P, \tilde{g}^* \circ \hat{P}) \right] \end{aligned} \tag{21}$$

We note that

$$\begin{aligned} &\mathrm{disc}_{\mathcal{V}}(g^* \circ P, \tilde{g}^* \circ \hat{P}) \\ &= \ \sup_{v \in \mathcal{V}} \left| \mathbb{E}_{z \sim g^* \circ P}[v(z)] - \mathbb{E}_{z \sim \tilde{g}^* \circ \hat{P}}[v(z)] \right| \\ &= \ \sup_{v \in \mathcal{V}} \left| \frac{1}{2} \sum_{c=1,2} \mathbb{E}_{z \sim g^* \circ P_c}[v(z)] - \frac{1}{2} \sum_{c=1,2} \mathbb{E}_{z \sim \tilde{g}^* \circ \hat{P}_c}[v(z)] \right| \\ &\leq \ \frac{1}{2} \sum_{c=1,2} \sup_{v \in \mathcal{V}} \left| \mathbb{E}_{z \sim g^* \circ P_c}[v(z)] - \mathbb{E}_{z \sim \tilde{g}^* \circ \hat{P}_c}[v(z)] \right| \\ &\leq \ \frac{1}{2} \sum_{c=1,2} \mathrm{disc}_{\mathcal{V}}(g^* \circ P_c, \tilde{g}^* \circ \hat{P}_c). \end{aligned} \tag{22}$$

Hence,

$$\mathbb{E}_{P_1 \neq P_2} \left[\mathrm{disc}_{\mathcal{V}}(g^* \circ P, \tilde{g}^* \circ \hat{P}) \right] \ \leq \ \mathbb{E}_{P_c \sim \mathcal{D}_1} \left[\mathrm{disc}_{\mathcal{V}}(g^* \circ P_c, \tilde{g}^* \circ \hat{P}_c) \right]. \tag{23}$$

Finally, by Proposition 5 in [5] we obtain that

$$L_{\hat{P}}(e_{\hat{P}} \circ f \circ \tilde{g}, \hat{y}) \ \leq \ \frac{16 V_{f \circ \tilde{g}}(\hat{P}_1, \hat{P}_2)}{s(f \circ \tilde{g}, \hat{P}_1, \hat{P}_2)}. \tag{24}$$

References

1. Ben-david, S., Blitzer, J., Crammer, K., Pereira, F.: Analysis of representations for domain adaptation. In: Advances in Neural Information Processing Systems 19, pp. 137–144. Curran Associates, Inc. (2006)
2. Benaim, S., Khaitov, M., Galanti, T., Wolf, L.: Domain intersection and domain difference. In: ICCV (2019)
3. Chazelle, B.: The discrepancy method - randomness and complexity. Cambridge University Press (2000)
4. Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., et al.: An image is worth 16x16 words: Transformers for image recognition at scale. arXiv preprint arXiv:2010.11929 (2020)
5. Galanti, T., György, A., Hutter, M.: On the role of neural collapse in transfer learning. In: International Conference on Learning Representations (2022), `https://openreview.net/forum?id=SwIp410B6aQ`
6. Goyal, A., Law, H., Liu, B., Newell, A., Deng, J.: Revisiting point cloud shape classification with a simple and effective baseline. arXiv preprint arXiv:2106.05304 (2021)
7. Huang, X., Liu, M.Y., Belongie, S., Kautz, J.: Multimodal unsupervised image-to-image translation. In: ECCV (2018)
8. Koltchinskii, V., Panchenko, D.: Rademacher processes and bounding the risk of function learning (2004)
9. Lasinger, K., Ranftl, R., Schindler, K., Koltun, V.: Towards robust monocular depth estimation: Mixing datasets for zero-shot cross-dataset transfer. CoRR **abs/1907.01341** (2019), `http://arxiv.org/abs/1907.01341`
10. Li, J., Selvaraju, R.R., Gotmare, A.D., Joty, S., Xiong, C., Hoi, S.: Align before fuse: Vision and language representation learning with momentum distillation. In: NeurIPS (2021)
11. Liu, Y., Fan, Q., Zhang, S., Dong, H., Funkhouser, T.A., Yi, L.: Contrastive multimodal fusion with tupleinfonce. 2021 IEEE/CVF International Conference on Computer Vision (ICCV) pp. 734–743 (2021)
12. Mansour, Y.: Learning and domain adaptation. In: Algorithmic Learning Theory, 20th International Conference, ALT. pp. 4–6 (2009)
13. Mansour, Y., Mohri, M., Rostamizadeh, A.: Domain adaptation: Learning bounds and algorithms. In: COLT - The 22nd Conference on Learning Theory (2009)
14. Mohri, M., Rostamizadeh, A., Talwalkar, A.: Foundations of Machine Learning. MIT Press, Cambridge, MA, 2 edn. (2018)
15. Müller, A.: Integral probability metrics and their generating classes of functions advances in applied probability. In: Advances in Applied Probability. pp. 429—443 (1997)
16. Papyan, V., Han, X.Y., Donoho, D.L.: Prevalence of neural collapse during the terminal phase of deep learning training. Proceedings of the National Academy of Sciences **117**(40), 24652–24663 (2020)
17. Qi, C.R., Yi, L., Su, H., Guibas, L.J.: Pointnet++: Deep hierarchical feature learning on point sets in a metric space. arXiv preprint arXiv:1706.02413 (2017)
18. Radford, A., Kim, J.W., Hallacy, C., Ramesh, A., Goh, G., Agarwal, S., Sastry, G., Askell, A., Mishkin, P., Clark, J., et al.: Learning transferable visual models from natural language supervision. In: International Conference on Machine Learning. pp. 8748–8763. PMLR (2021)

19. Reed, S., Akata, Z., Yan, X., Logeswaran, L., Schiele, B., Lee, H.: Generative adversarial text to image synthesis. In: Balcan, M.F., Weinberger, K.Q. (eds.) Proceedings of The 33rd International Conference on Machine Learning. Proceedings of Machine Learning Research, vol. 48, pp. 1060–1069. PMLR, New York, New York, USA (20–22 Jun 2016)
20. Ridnik, T., Ben-Baruch, E., Noy, A., Zelnik-Manor, L.: Imagenet-21k pretraining for the masses (2021)
21. Sun, K., Xiao, B., Liu, D., Wang, J.: Deep high-resolution representation learning for human pose estimation. In: CVPR (2019)
22. Xu, K., Ba, J., Kiros, R., Cho, K., Courville, A., Salakhudinov, R., Zemel, R., Bengio, Y.: Show, attend and tell: Neural image caption generation with visual attention. In: Bach, F., Blei, D. (eds.) Proceedings of the 32nd International Conference on Machine Learning. Proceedings of Machine Learning Research, vol. 37, pp. 2048–2057. PMLR, Lille, France (07–09 Jul 2015)
23. Zhang, R., Guo, Z., Zhang, W., Li, K., Miao, X., Cui, B., Qiao, Y., Gao, P., Li, H.: Pointclip: Point cloud understanding by clip. arXiv preprint arXiv:2112.02413 (2021)
24. Zhu, J.Y., Park, T., Isola, P., Efros, A.A.: Unpaired image-to-image translation using cycle-consistent adversarial networks. 2017 IEEE International Conference on Computer Vision (ICCV) pp. 2242–2251 (2017)