Contrastive Feature Learning for Audio Classification

Daniel Lin

Acknowledgement

I would like to thank my advisor, Professor Stella Yu, on her guidance and for giving me the opportunity to learn how to conduct meaningful research. I would also especially like to thank Dr. Sascha Hornauer for being a great mentor and devoting countless hours and meetings to helping me improve my work. I would also like thank Dr. Shabnam Ghaffarzadegan and Bosch for guiding and introducing me to industry related research.

Also, I would love to shout out all my roommates and friends who were there for me throughout my five years at Berkeley; I would not have made it here without them. Lastly and most importantly, I would like to thank my family for their unconditional love and support.

Contrastive Feature Learning for Audio Classification

By Daniel Lin

Research Project

Submitted to the Department of Electrical Engineering and Computer Sciences, University of California at Berkeley, in partial satisfaction of the requirements for the degree of **Master of Science, Plan II**.

Approval for the Report and Comprehensive Examination:

Committee:

[illegible]

Professor Stella Yu
Research Advisor

5/14/2022

(Date)

* * * * * * *

[illegible]

Professor Michael Lustig
Second Reader

5/12/2022

(Date)

Contrastive Feature Learning for Audio Classification

by

Daniel Lin

A thesis submitted in partial satisfaction of the

requirements for the degree of

Master of Science

in

Electrical Engineering and Computer Science

in the

Graduate Division

of the

University of California, Berkeley

Committee in charge:

Professor Stella Yu, Chair
Professor Michael Lustig

Spring 2022

Contrastive Feature Learning for Audio Classification

Abstract

Contrastive Feature Learning for Audio Classification

by

Daniel Lin

Master of Science in Electrical Engineering and Computer Science

University of California, Berkeley

Professor Stella Yu, Chair

Models currently can accurately classify images if given a large labeled dataset. However, labeling is often a tedious process and does not scale well. Contrastive learning aims to learn an embedding space in which similar pairs are mapped close to each other and dissimilar pairs are mapped away from each other. This self-supervised learning method can leverage unlabeled datasets to generate feature representations that can be applied to various tasks such as classification and segmentation. Most of the current literature in contrastive learning deals with images, but there are significantly fewer works related to audio inputs. Recently, deep learning showed promising results to solve tasks such as event classification on audio data, so I believe some of the contrastive learning techniques can be applied to this domain as well. In this thesis, I investigate the effectiveness of two different instance discrimination frameworks, non-parametric instance discrimination (NPID) and Momentum Contrast (MoCo) that are trained on audio event data. I demonstrate that a network, without large amounts of data, can learn an audio representation that can be applied to improve performance on various classification tasks, including music and environmental sounds. Finally, my experiments show that pretrained weights from these frameworks can lead to faster convergence than other standard weight initialization methods.

To my Mom, Dad, and Sister

Contents

Contents **ii**

List of Figures **iii**

List of Tables **iv**

1 Introduction **1**
1.1 Self-Supervised Representation Learning 1
1.2 Contrastive Learning in Computer Vision 3
1.3 Audio Representation 12
1.4 Contrastive Learning in Audio 16

2 Experiments **17**
2.1 Experimental Setup 17
2.2 Generating Feature Representations 19
2.3 Evaluating Feature Representations 20

3 Results **21**
3.1 Feature Representation Generation 21
3.2 Classification 22

4 Conclusions **31**

Bibliography **32**

List of Figures

1.1 Predicting which rotation is applied. Image Source: [12] 2
1.2 The transformation of image a) Original image with tiling b) Shuffling the tiles around c) Predicting the correct output tile order. Image Source: [23] 2
1.3 Examples of a grayscale input image and colorized output images. Image Source: [37] 3
1.4 Comparison between Contrastive, Triplet, and Lifted Structure Embeddings. Image Source: [31] 4
1.5 Training Pipeline of NPID. Image Source: [36] 7
1.6 Objectives of CLD. Image Source: [35] 8
1.7 Standard Pipeline of Siamese Networks 9
1.8 SimCLR pseudocode and loss function. Image Source: [4] 9
1.9 MoCo Training Pipeline. Image Source: [17] 10
1.10 SwAV training Pipeline. Image adapted from [3] 12
1.11 Individual Spectrogram Augmentations. From the top to bottom: original, time-warped, frequency-masked, and time-masked spectrogram. Image source: [25] . 14
1.12 Pitch Scale vs Mel Scale. Image Source: [19] 15
1.13 Mel Spectrogram vs Wavelet Transform. Image adapted from [1] 15
1.14 COLA training pipeline. Image Source: [27] 16

2.1 Experimental Pipeline. The experiments are run in two stages, pretraining and evaluation. A network learns a representation during pretraining and then uses that representation in a downstream classification task during evaluation. 17
2.2 Fine-Tuning vs Freezing Experiments 19

3.1 NPID Loss 21
3.2 MoCo Loss 21
3.3 ESC-50 Validation Curves 23
3.4 US8K Validation Curves 24
3.5 TSNEs for True and Predicted Labels for each type of network on US8K Fold 1 26
3.6 Heatmaps for Each Type of Network on US8K Fold 1 27
3.7 DCASE 2013 Validation Curves 29
3.8 GTZAN Validation Curves 30

List of Tables

1.1 Summary of Contrastive Learning Method and Loss Used 13

3.1 Best Average Validation Accuracies ESC 50 23
3.2 Best Average Validation Accuracies US8K 24
3.3 Class-wise F1 Scores of US8K Fold 1 for different classification methods 28
3.4 Best Average Validation Accuracies DCASE13 29
3.5 Test Accuracies GTZAN 30

Acknowledgments

I would like to thank my advisor, Professor Stella Yu, on her guidance and for giving me the opportunity to learn how to conduct meaningful research. I would also especially like to thank Dr. Sascha Hornauer for being a great mentor and devoting countless hours and meetings to helping me improve my work. I would also like thank Dr. Shabnam Ghaffarzadegan and Bosch for guiding and introducing me to industry related research.

Also, I would love to shout out all my roommates and friends who were there for me throughout my five years at Berkeley; I would not have made it here without them. Lastly and most importantly, I would like to thank my family for their unconditional love and support.

Chapter 1

Introduction

1.1 Self-Supervised Representation Learning

Fully supervised learning, like a task similar to classifying images from ImageNet [8], can sometimes necessitate using large amounts of labeled data to generate representations. Thus, researchers have been recently looking more into self-supervised learning, which may be a potential technique to generate similar representations with a significantly smaller amount of labeled data. Self-supervised learning creates a pretext task from the data itself, and then a network is trained to solve that pretext task. By completing the pretext task, a network aims to learn representations that can be used to solve a future downstream task. Recent work has shown that the learned representations from solving a pretext task can be transferred to do well in other supervised learning tasks [20]. I show some examples of three separate image-based pretext tasks below.

Rotation

Gidaris et al. [12] introduced a cheap way to learn feature representations without major augmentations through an image rotation pretext task. The authors took example images, rotated them 4 different ways, and trained a network on a simple classification task to predict the orientation of the image, as shown in Figure 1.1. In order to solve the task, a network had to learn about the relative locations of different parts of an object, such as facial features, legs, leaves, etc. The authors were able to train a linear classifier on top of the learned representations and showed similar results to direct supervised feature learning.

Jigsaw

Noroozi and Favaro [23] introduced a different pretext task, solving a jigsaw puzzle. In the jigsaw puzzle, a network is trained to place nine separate tiles of an image back into their original location, shown in Figure 1.2. To prevent the number of outputs being on the order of $O(K!)$ where K is the number of jigsaw pieces, a network is trained to predict

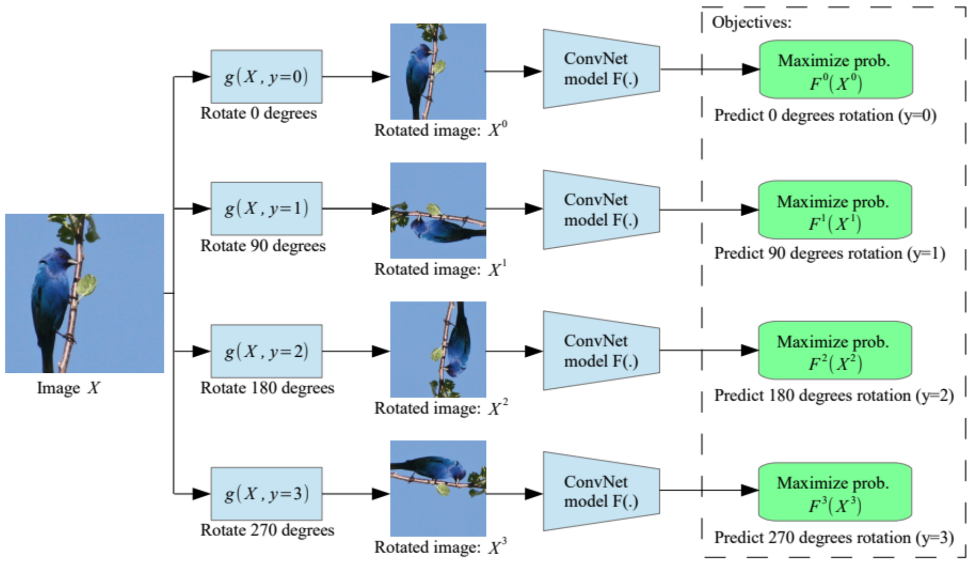

Objectives:
Image X
$g(X, y=0)$
Rotate 0 degrees
$g(X, y=1)$
Rotate 90 degrees
$g(X, y=2)$
Rotate 180 degrees
$g(X, y=3)$
Rotate 270 degrees
Rotated image: X^0
Rotated image: X^1
Rotated image: X^2
Rotated image: X^3
ConvNet model F(.)
Maximize prob. $F^0(X^0)$
Predict 0 degrees rotation (y=0)
Maximize prob. $F^1(X^1)$
Predict 90 degrees rotation (y=1)
Maximize prob. $F^2(X^2)$
Predict 180 degrees rotation (y=2)
Maximize prob. $F^3(X^3)$
Predict 270 degrees rotation (y=3)

Figure 1.1: Predicting which rotation is applied. Image Source: [12]

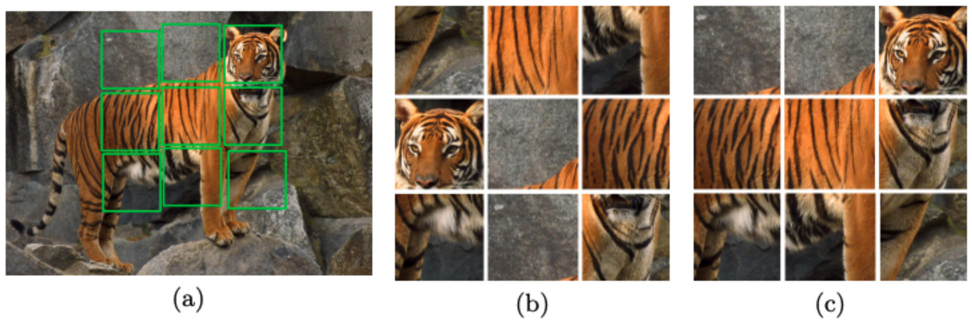


Figure 1.2: The transformation of image a) Original image with tiling b) Shuffling the tiles around c) Predicting the correct output tile order. Image Source: [23]

the correct order out of a fixed set of permutations. This allows the authors to control the difficulty by setting the size of the set of possible permutations. The authors argued that solving the Jigsaw puzzle task can help a network learn about the different parts of an object and understand relative positions of features within an object.

Colorization

Zhang et. al [37] analyzed the task of colorizing a grayscale image. The authors approached the colorization problem in a few unique ways. First, they used the CIE color space instead of the RGB space. The CIE color space more mimics human vision, while

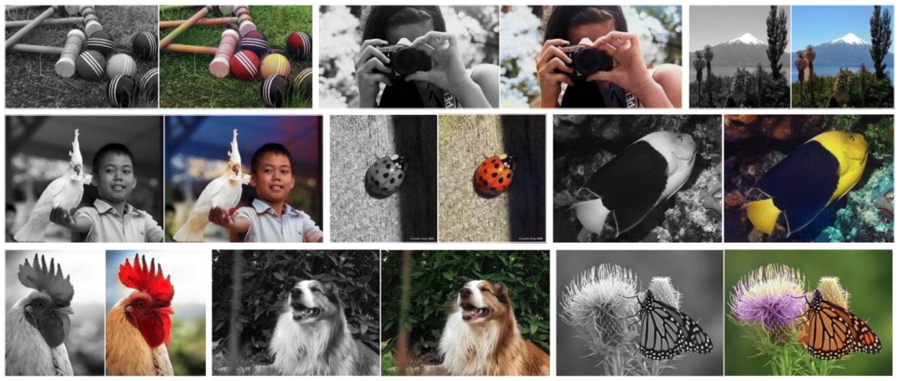


Figure 1.3: Examples of a grayscale input image and colorized output images. Image Source: [37]

the RGB space is more conducive to how a camera sees objects. Also, the authors used a cross-entropy loss of the predicted color distribution over binned values instead of a standard regression-based pixel by pixel L2-loss because they believed that the output images from networks trained with an L2-loss often looked desaturated. The authors found that even though the colorization task feels like more of a graphics problem rather than a vision problem, the representations learned are surprisingly useful for classification, detection, and segmentation.

1.2 Contrastive Learning in Computer Vision

One prominent type of self-supervised representation learning in computer vision is called contrastive learning. In this type of learning, a network tries to cluster like images together and repel different images. The thought process behind this is that humans are able to associate similar objects together without the explicit need to label each image's individual class. Contrastive learning in computer vision was first used to try and distinguish between faces [6], but has since been extended to learning representations of various types of images. Many of the networks today can use contrastive learning techniques on a small fraction of ImageNet [8] data to generate quality representations. These representations only need slight fine-tuning on most image-based supervised datasets to achieve similar or greater performance than direct supervised learning on the given dataset. [17, 16, 4, 5, 35]

Loss Functions

Prediction problems often try to minimize an objective function. Contrastive learning is fundamentally about attracting like instances and repelling different ones. Thus, most of the losses either try to maximize the similarity between like instances or maximize the probability of picking a like instance given a sample.

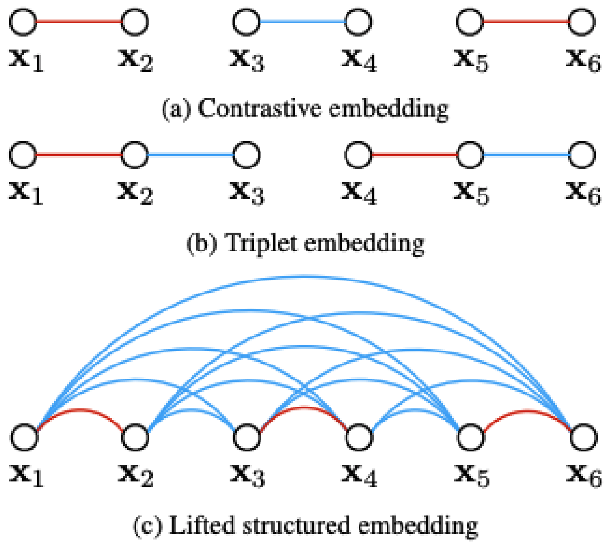


Figure 1.4: Comparison between Contrastive, Triplet, and Lifted Structure Embeddings. Image Source: [31]

Contrastive Loss

Chopra et al. [6] first introduced the contrastive loss when trying to discriminate between faces. The contrastive loss may be shown as

$$L(x_i, x_j) = \mathbb{1}[y_i = y_j] L_S\left(f(x_i), f(x_j)\right) + \mathbb{1}[y_i \neq y_j] L_D\left(f(x_i), f(x_j)\right)$$

where L_S is the loss function for two images in the same class, and L_D is the loss function for two images in different classes. When looking at the loss functions, L_S should be decreasing as $f(x_i)$ becomes more "closer" in features space as $f(x_j)$ because x_i and x_j belong to the same class. However, L_D should be increasing as $f(x_i), f(x_j)$ become "closer" in the feature space because we want to have different representations for data points in different classes. An example of a L_D can be of the form

$$L_D = \max(0, \epsilon - L_S)$$

where ϵ can be a tune able parameter of the maximum divergence between the feature representation and L_S can be Euclidean distance, a function of cosine similarity, or another way of showing divergence between two vectors.

Triplet Loss

Following contrastive loss, Schroff et al. [29] introduced a different type of loss called triplet loss with a similar motivation of facial recognition. The authors take a sample anchor point, a positive sample, and a negative sample. The positive sample is of the same class as the anchor point, and the negative sample is of a different class. This framework aims to minimize the distance between the representation of the positive sample and the anchor while maximizing the distance between the anchor and the negative sample. This loss can defined as

$$L(x_a, x_p, x_n) = \max\big(0, D(x_a, x_p) - D(x_a, x_n) + \epsilon\big)$$

where $D(x_i, x_j)$ is some distance metric like L2 distance and ϵ is a margin parameter.

Lifted Structure Loss

Both of the previous losses, contrastive and triplet loss, primarily use $O(B)$ pairs per batch and just compare singular pairs within a batch during training. Song et al. [31] thought it would be more useful to consider all $O(B^2)$ edges in training, as shown in Figure 1.4. Their loss function is then

$$L = \frac{1}{2|P|} \sum_{i,j \in P} \big(\max(0, L_{ij})\big)^2$$

$$L_{ij} = \max\left(\max_{i,k \in N}(\epsilon - D_{ik}, 0), \max_{j,k \in N}(\epsilon - D_{jk}, 0)\right) + D_{ij}$$

where P and N are the positive and negative pairs in a batch respectively, D_{ij} is a distance metric for the representations of the images in a batch, and ϵ is a maximum error threshold. This loss function tries to minimize the average of the maximum divergence of negative samples from positive ones. L_{ij} is not smooth because of all of the max functions, so the loss was smoothed to

$$\tilde{L_{ij}} = \log\left(\sum_{i,k \in N} \exp(\epsilon - D_{ik}) + \sum_{j,k \in N} \exp(\epsilon - D_{jk})\right)$$

N-pair Loss

Triplet loss only considers 1 positive and 1 negative sample along with an anchor, so Sohn [30] decided to extend the number of negative samples to N - 1 instead of just 1. The philosophy behind using more samples is that in triplet loss, a network is trying to differentiate a sample from only one negative class. However, in N-pair loss, all the negative classes in a batch are being used. Thus, the loss becomes

$$L(x_a, x_p, x_{ni}) = \max\big(0, D(x_a, x_p) - D(x_a, x_n) + \epsilon\big)$$

However, like lifted structured loss, this function is good for negative mining but is not smooth and does not work too well in practice. Thus, it is smoothed out to a somewhat log likelihood function so that

$$L(x_a, x_p, x_{ni}) = -\log \frac{\exp(D(x, x_p))}{\exp\left(D(x, x_p)) + \sum_i \exp(D(x, x_{ni})\right)}$$

Noise Contrastive Estimation (NCE)

The idea behind Noise Constrastive Estimation (NCE) [13] is to distinguish noise samples from data points using logistic regression. Consider data points $x_i|C = 1 \sim p_d$ and noise $y_j|C = 0 \sim p_n$ where p_d and p_n are distributions of data points and noise respectively. Then, given a random sample point u, the likelihood of u being labeled as data is

$$P(C = 1|u) = \frac{p_d(\mu; \theta)}{p_d(\mu; \theta) + kp_n(\mu)}$$

where k is a hyper-parameter computing the ratio of noise to data points per iteration. The full objective is then to minimize the negative log posterior.

$$L(x_1 \cdots x_n, y_1 \cdots y_m) = \frac{1}{n} \sum_{i=1}^{n} \log\left(P(C = 1|x_i)\right) + k * \frac{1}{m} \sum_{j=1}^{m} \log\left(1 - P(C = 1|y_j)\right)$$

where x_i and y_j are data points and noise respectively.

InfoNCE

InfoNCE loss [24], inspired from NCE, uses the standard cross-entropy loss in supervised classification to distinguish a positive sample from negative noise samples. From an anchor x_a, one positive sample from the conditional distribution $p(x|x_a)$ and $N - 1$ samples from the proposal distribution $p(x)$ are drawn. Thus, the probability a sample is chosen is

$$p(x_p|X = x_{ni}, x_a) \propto \frac{f(x_p, x_a)}{f(x_p, x_a) + \sum_i f(x_{ni}, x_a)}$$

The InfoNCE loss just minimizes the negative log probability, giving

$$L(x_p, x_a, x_{ni}) = -\mathbb{E}\left[\log \frac{f(x_p, x_a)}{f(x_p, x_a) + \sum_i f(x_{ni}, x_a)}\right]$$

Memory Bank Models

One way of keeping track of negative samples is by having a memory bank. This memory bank stores individual instances and is thus is $O(N)$ space complexity with respect to number of data points. In most cases, the loss involves predicting the most likely label of a sample point given its representation and is often softmax-based. Thus, the memory bank of stored representations helps to avoid recomputing the denominator each time and allows for faster calculation of the conditional probability of the positive sample given an anchor.

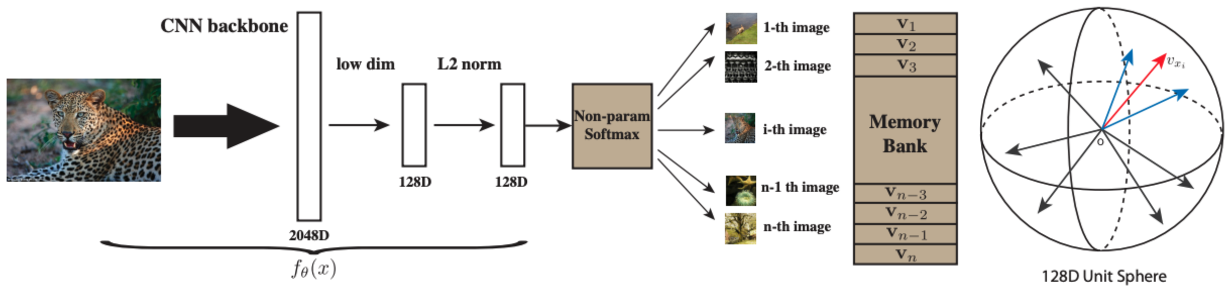


Figure 1.5: Training Pipeline of NPID. Image Source: [36]

NPID

Non-Parametric Instance Discrimination (NPID) [36] attempts to learn a good feature representation without the necessity of labeled classes. The motivation behind this approach is that babies can see similarities between objects before they learn language; thus there should be a way to create representations of individual objects without labels. As shown in Figure 1.5, the authors first encode each feature representation $v = f_\theta(x)$ and project it onto an N-dimensional hyper-sphere where $||v||^2 = 1$. Then, they calculate the standard softmax probability of a feature representation v correspoding to the i-th sample

$$P(i|v) = \frac{\exp(v_i^T v/\tau)}{\sum_i \exp(v_i^T v/\tau)}$$

where τ is a tune-able temperature parameter of the confidence of the softmax. However, the size of the softmax is $O(N)$, as each instance is treated as its own class. Thus, computing the full softmax is very costly, so the authors adapt NCE to turn the problem into a binary classification problem of data vs noise. Thus, their final objective (prior to some additional optimization and regularization for computational purposes) looks like

$$L_{NCE} = -\mathbb{E}_{P_d}\left[\log h(i,v)\right] - m * \mathbb{E}_{P_n}\left[\log\left(1 - h(i,v')\right)\right]$$

where P_d is the true data distribution, P_n is the noise distribution, $v = f(x_i)$ and $v' = f(x_j)$ where $i \neq j$. The posterior distribution $h(i,v)$ of the data distribution $D = 1$ containing a sample i and its corresponding to feature v is defined as

$$h(i,v) = P(D = 1|i,v) = \frac{P(i|v)}{P(i|v) + m * P_n(i)}$$

where m is an assumption of how frequently noise samples appear when compared with data samples.

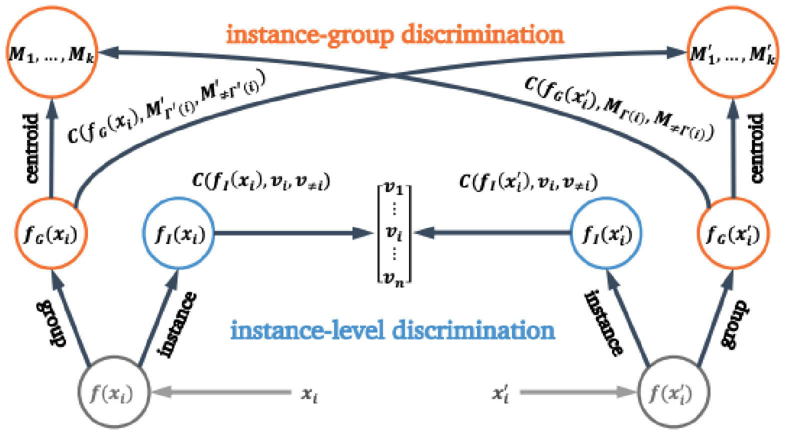

instance-group discrimination
$M_1, \dots, M_k$
$M'_1, \dots, M'_k$
$C(f_G(x_i), M'_{\Gamma'(i)}, M'_{\neq\Gamma'(i)})$
$C(f_G(x'_i), M_{\Gamma(i)}, M_{\neq\Gamma(i)})$
centroid
centroid
$C(f_I(x_i), v_i, v_{\neq i})$
$C(f_I(x'_i), v_i, v_{\neq i})$
$f_G(x_i)$
$f_I(x_i)$
$v_1 \dots v_i \dots v_n$
$f_I(x'_i)$
$f_G(x'_i)$
group
instance
instance-level discrimination
instance
group
$f(x_i)$
x_i
x'_i
$f(x'_i)$

Figure 1.6: Objectives of CLD. Image Source: [35]

CLD

NPID treats every individual image as its own class, so sometimes it does not take advantage of the relative similarity between images in the same class. Cross Level Discrimination[35], instead of discriminating between all images, only discriminates between images within a class while trying to keep class clusters separate. Let a constrastive loss function $C(f, f^+, f^-, \tau)$ be defined as

$$C(f, f^+, f^-, \tau) = -\log\left(\frac{\frac{\exp<f,f^+>}{\tau}}{\frac{\exp<f,f^+>}{\tau} + \sum \frac{\exp<f,f^-)>}{\tau}}\right)$$

where f, f^+, and f^- are feature representations of an instance, positive sample, and negative sample respectively, and τ is a temperature parameter. As shown in Figure 1.6 the network has two forks, an instance branch f_i and a group branch f_g. In the instance branch, the authors compare instances with the global memory bank v_i with an InfoNCE type loss. Thus, the loss for an instance is just

$$L_{instance}(x_i, v_i) = C\left(f_i(x_i), v_i, v_{-i}, \tau_{instance}\right)$$

In the group branch f_g, the authors perform a local clustering to find different centroids M_j that we treat as separate "classes", and use cross-level discrimination to differentiate between the two centroids. The cross-level discrimination is then formulated as

$$L_{group}(x_i) = C\left(f_g(x_i), M_{\mathcal{T}(i)}, M_{-\mathcal{T}(i)}, \tau_{group}\right)$$

where $\mathcal{T}(i) = j$ if x_i belongs to cluster M_j.

So, the overall loss for a set of points x_i, x'_i in CLD is

$$L(x_i, x'_i) = L_{instance}(x_i, v_i) + L_{instance}(x'_i, v_i) + \lambda(L_{group}(x_i) + L_{group}(x'_i))$$

where λ is a weight parameter determining the weight of instance vs group discrimination. The 4 loss components are shown in Figure 1.6.

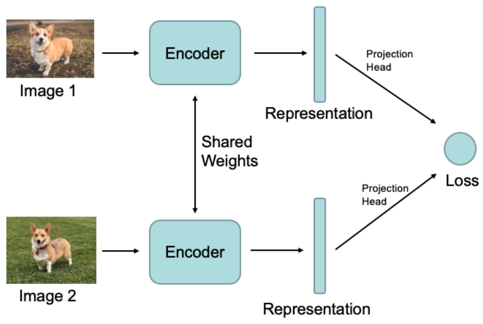
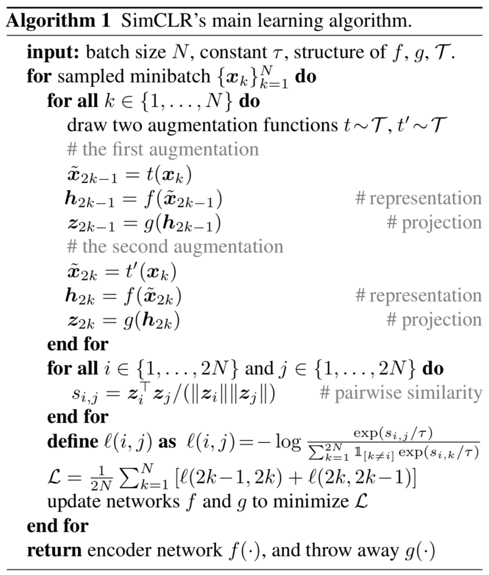


Figure 1.7: Standard Pipeline of Siamese Networks



```
Algorithm 1 SimCLR's main learning algorithm.
  input: batch size $N$, constant $\tau$, structure of $f$, $g$, $\mathcal{T}$.
  for sampled minibatch $\{\boldsymbol{x}_k\}_{k=1}^{N}$ do
    for all $k \in \{1, \ldots, N\}$ do
      draw two augmentation functions $t \sim \mathcal{T}$, $t' \sim \mathcal{T}$
      # the first augmentation
      $\tilde{\boldsymbol{x}}_{2k-1} = t(\boldsymbol{x}_k)$
      $\boldsymbol{h}_{2k-1} = f(\tilde{\boldsymbol{x}}_{2k-1})$                # representation
      $\boldsymbol{z}_{2k-1} = g(\boldsymbol{h}_{2k-1})$                # projection
      # the second augmentation
      $\tilde{\boldsymbol{x}}_{2k} = t'(\boldsymbol{x}_k)$
      $\boldsymbol{h}_{2k} = f(\tilde{\boldsymbol{x}}_{2k})$                # representation
      $\boldsymbol{z}_{2k} = g(\boldsymbol{h}_{2k})$                # projection
    end for
    for all $i \in \{1, \ldots, 2N\}$ and $j \in \{1, \ldots, 2N\}$ do
      $s_{i,j} = \boldsymbol{z}_i^\top \boldsymbol{z}_j / (\|\boldsymbol{z}_i\| \|\boldsymbol{z}_j\|)$        # pairwise similarity
    end for
    define $\ell(i,j)$ as $\ell(i,j) = -\log \frac{\exp(s_{i,j}/\tau)}{\sum_{k=1}^{2N} \mathbb{1}_{[k \neq i]} \exp(s_{i,k}/\tau)}$
    $\mathcal{L} = \frac{1}{2N} \sum_{k=1}^{N} [\ell(2k-1, 2k) + \ell(2k, 2k-1)]$
    update networks $f$ and $g$ to minimize $\mathcal{L}$
  end for
  return encoder network $f(\cdot)$, and throw away $g(\cdot)$
```

Figure 1.8: SimCLR pseudocode and loss function. Image Source: [4]

Siamese Network Baselines

Siamese Networks are network architectures with two separate branches. Usually, the two encoders share weights, but it is not necessary to do so. A standard pipeline is shown in Figure 1.7. There are two images, sometimes two augmentations of the same image, that are fed through an encoder. This encoder is usually some common image processing network like a ResNet [15], as this type of network is easier to benchmark against other methods in a supervised learning environment. At the end of the encoder there is sometimes a projection head to help map the features onto a lower dimension space, but this is usually replaced with a classifier when transferring weights to a separate task. The two representations are then compared with one of the above loss functions, and the gradients are propagated back to the encoders.

SimCLR

SimCLR [4](Simple Framework for Contrastive Learning of Visual Representations) is one of the most prominent Siamese networks. In SimCLR, the first step, given a sample batch of N points, is to make 2 augmentations per point, giving us 2N transformed points. Using one positive pair as an anchor and a positive point, we consider the rest of the 2N - 1 transformed points as negative samples and feed them through the Siamese network encoder. We then minimize the N-pair loss over the batch and update.

The full pseudocode and Loss function is shown in Figure 1.8. Later on, Chen et al. [5]

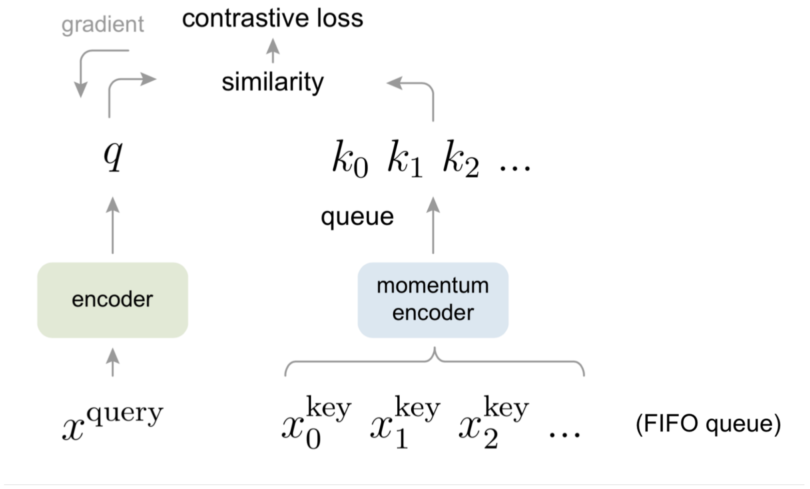


Figure 1.9: MoCo Training Pipeline. Image Source: [17]

improved on SimCLR to release SimCLR v2 with the following design decisions.

1) They increased the size of the encoder from a ResNet 50 to a ResNet 152
2) They increased the projection head depth from a 2 layer to 3 layer MLP
3) Inspired from MoCo [17], they incorporated the memory bank mechanism to buffer the negative samples.

The authors showed that the deeper networks significantly improved the representation of the image. Both networks are originally trained on images from ImageNet [8], and then their encoders are fine-tuned in standard supervised learning on 1% of the dataset. SimCLR v2 showed an increase from 58% to 74% on ImageNet overall top 1% accuracy.

MoCo

MoCo (Momentum Contrast) [17] was a new idea that did not have an identical encoder on both sides of a Siamese network. Instead, it uses a momentum encoder on one side of the network, as shown in Figure 1.9. It also uses a mechanism similar to the standard memory bank of feature representations. However, a standard memory bank is inconsistent because a network is learning throughout the epoch. Thus, a representation of an image input stored in the memory bank at an earlier iteration of the epoch would be different than the representation of the same input at a later iteration of the epoch. MoCo tries to address this problem through using a momentum encoder. Samples are first queried as anchors x^{query} and get an encoded representation $q = f_{encoder}(x)$. Keys x^{key} are then created by using x^{query} as a base and adding another augmentation. As shown in Figure 1.9, during training, the current batch of images are used to create positive pairs, and a combination of queries from the current batch and keys from the previous batch are used to create negative pairs. The objective is then created by minimizing the temperature controlled InfoNCE loss

$$L = -\log \frac{\exp(q * k_+/\tau)}{\sum \exp(q * k_i \tau)}$$

MoCo can be used with smaller batches of 256 instead of large batches like in SimCLR 4096 because MoCo does not need to generate a large amount of negative samples per training step. Also in SimCLR, the two encoders in the Siamese network are identical, but MoCo has a momentum encoder whose parameters contain a momentum coefficient $m \in [0, 1)$ to incorporate both the encoder and the previous state of the momentum encoder. The momentum encoder is defined as

$$\theta_k = m * \theta_k + (1 - m) * \theta_q$$

where θ_k and θ_q are the parameters of the momentum encoder and encoder respectively, with only θ_q being updated by back-propagation. The reason why this is important is because we want to keep the representations of the two different encoders as consistent as possible because we are querying keys from a previous iteration. Memory banks keep inconsistent representations because they store the representation, not the image or key itself. Thus, MoCo converting keys at training time updates representations each time.

He et al. [16] then improved the MoCo performance with a few ideas from SimCLR. Originally, MoCo only had 1 FC layer at the end of the encoder, but they modified it to a 2 layer projection head, and the authors also included some more data augmentations like blurring to the training.

SwAV

Another way of trying to contrastively learn representations of images is through clustering. Traditional methods like deep-cluster [2] propose methods in two steps and two passes over the dataset. During the first step, the network clusters the image features offline, and during the second step, the network uses those previous clusters as "classes" and tries to predict those classes. However, this method is both slow and prone to collapsing to trivial solutions.

Instead, Caron et al. [3] use a method called SwAV (Swapping Assignments between multiple Views of the same image). Unlike deep-cluster's [2] offline clustering and direct comparison of features, SwAV is able to simultaneously cluster data while enforcing consistency between different augmentations. The full pipeline is shown in Figure 1.10. Each image x is first augmented and transformed into two separate images, x_1 and x_2. Then, this augmented view is mapped through f_θ, a ResNet [15] encoder with a MLP projection head on top into normalized features z_1 and z_2. These features are then mapped onto a set of K prototype vectors, which are essentially cluster centers, into codes q_1 and q_2. At the end, the

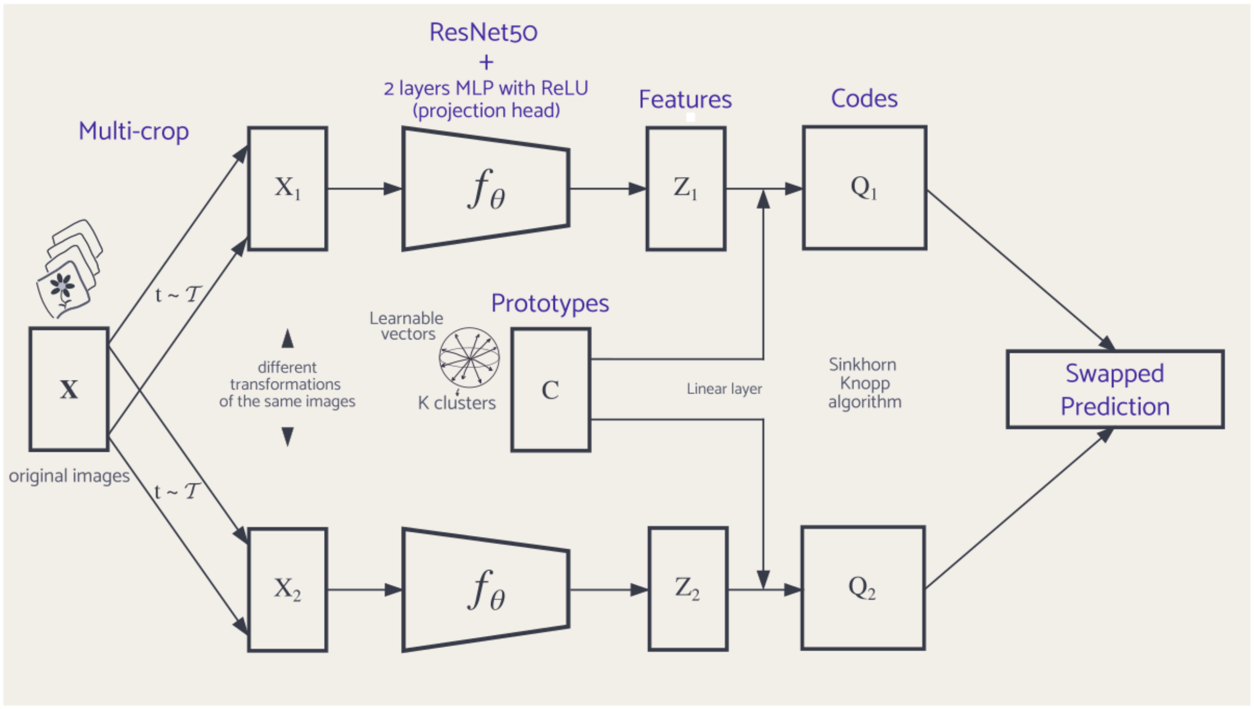


Figure 1.10: SwAV training Pipeline. Image adapted from [3]

authors use a small swap to get the following loss

$$L(z_1, z_2) = \ell(z_1, q_2) + \ell(z_2, q_1)$$

where l is a measure of fit between a feature vector and a code. The intuition behind this swap is that two augmented images showing the same subject should be able to predict the code from the other augmented image. Each measure of fit is a cross-entropy loss between the code q_k and the softmax probability of obtaining the dot product of feature z_t and prototype vectors c_k

$$\ell(q_s, z_t) = -\sum_k q_s^{(k)} \log p_t^{(k)} \text{where } p_t^{(k)} = \frac{\exp(z_t^\top c_k)/\tau}{\sum_{k'} \exp(z_t^\top c_{k'})/\tau}$$

1.3 Audio Representation

Sound is a signal that propagates through air (or other mediums) as a longitudinal wave, a type of wave that moves in the same direction that it started at. Each sound wave consists of vibrations moving away from a specific source and is usually represented by measurements through time and channels. The measurements taken are the vibrations in the air and are taken at a frequency called the sampling rate, which is the number of

Method	Loss Functions Used
NPID	NCE
CLD	InfoNCE
SimCLR	N-Pair
MoCo	InfoNCE
SwAV	Cross Entropy

Table 1.1: Summary of Contrastive Learning Method and Loss Used

measurements per second. Each channel represents sound coming from or going to a single point. For example, humans have two ears, so most sounds we hear is the aggregation of two separate channels. Audio data can also be augmented through various transformations, including shifting pitch, adding noise, and changing speed. Also, audio signals can often be directly overlayed on top of each other such that a listener would be able to decipher the resulting waveform as two separate signals merged into one [10]. Even though waveforms show much information about an audio signal, most literature convert these waveforms into an image-like input. A raw waveform only shows information with time on the x-axis and magnitude on the y-axis. However, a visual input like a spectrogram maps the waveform into a different domain that includes frequency on the y-axis. The resulting input has information includes information about time, magnitude, and frequencies found. Processing image-like inputs have additionally progressed far more than processing raw waveforms, so it makes sense to take advantage of architectures shown in computer vision. Below are some common transformations from raw auditory signal to visual input.

Spectrograms

A spectrogram is a visual representation of an audio signal through use of the Short Time Fourier Transform (STFT). The discrete-time STFT with an L-point window can be written as

$$X[m,n] = \sum_{k=0}^{L-1} x[k]g[k-m]e^{-i2\pi \frac{kn}{L}}$$

with $x[k]$ denoting a signal, $g[k-m]$ denoting a window function. Each STFT pass goes over a specific continuous subsection of a signal, so window functions are used to denote how the signal is convolved within the section. This is important because a basic uniform rectangular window causes discontinuities at the boundaries. Most of the windows used follow a more smooth triangular or bell-curve like shape, like the Hamming or Blackman Window.

The spectrogram is often constructed on the log-spectrum $\text{Spec}(m,n) = 20 \log |X[m,n]|$ because this visualization gives the spectra in decibels. It is represented with time on the X-

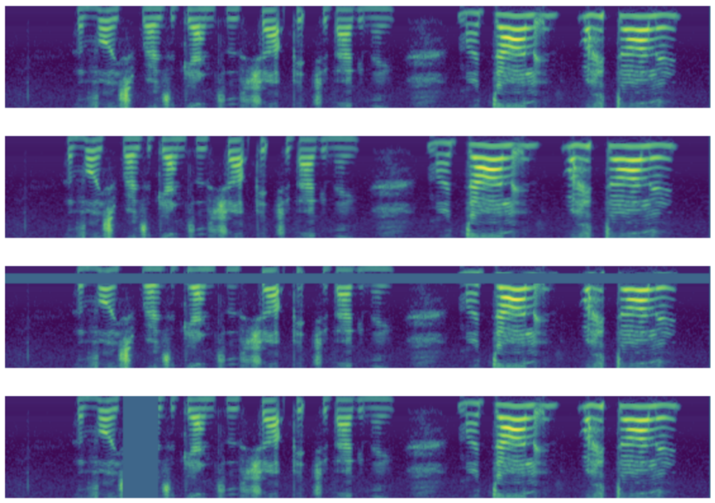


Figure 1.11: Individual Spectrogram Augmentations. From the top to bottom: original, time-warped, frequency-masked, and time-masked spectrogram. Image source: [25]

axis, frequency on the Y-axis, and amplitude represented by color, i.e darker colors represent a stronger signal and lighter ones a weaker signal.

In a spectrogram, there is a tradeoff between the time and frequency resolution. This relates to the Heisenberg uncertainty principle and Gabor limit, which states that one cannot figure out the exact time and frequency of a signal simultaneously. For example, if the window-size is increased, then the frequency resolution would increase and the time resolution would decrease. In this case, there is more signal being captured in a window, resulting in more frequencies being detected.

Similar to image transforms, spectrograms have their own set of transforms, introduced by Park et al. [25] The standard rotation, flipping, color jitter, etc. do not make too much sense to capture different audio signals. Instead, there are time-masking, frequency masking, and time-warping and compositions of those. Individually, they are shown in Figure 1.11.

Mel Spectrogram

A commonly used variation of the spectrogram is the Mel Spectrogram. The motivation behind this transform is that humans perceive frequencies on a non-linear scale. For example, a human can detect changes a lot easier between 1000 Hz and 2000 Hz relative to 10000 Hz and 11000 Hz. The Mel Scale is a scale that converts frequency into a unit such that equidistant measures on the mel-scale represent an equidistant, human judged, difference in tones, as shown in Figure 1.12. Some take the Mel Spectrogram a step further and use the Mel-Frequency Cepstral Coefficients (MFCC), which is computed by taking the Discrete Cosine Transform (DCT) on the log Mel Spectrogram. The main advantage of the MFCC is that it is more compressible and more decorrelated compared to the Mel spectrogram.

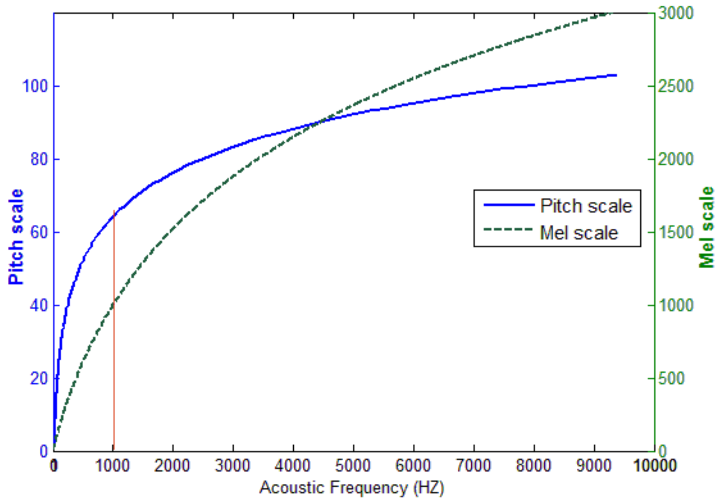


Figure 1.12: Pitch Scale vs Mel Scale. Image Source: [19]

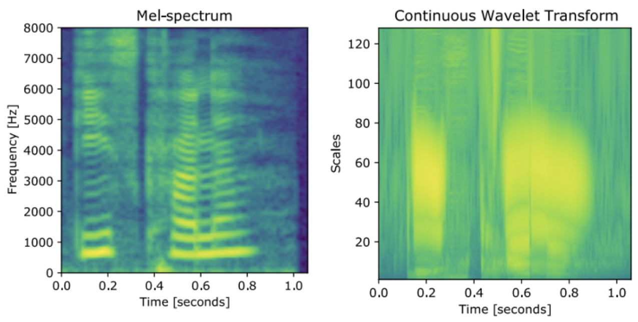


Figure 1.13: Mel Spectrogram vs Wavelet Transform. Image adapted from [1]

Wavelet Transform

The wavelet transform is a different visualization of an audio signal. Instead of using a fixed window size and the Fourier Transform, the wavelet transform makes use of a mother wavelet. A continuous wavelet transform (CWT) is defined as

$$\mathrm{CWT}_x(u,t) = \frac{1}{\sqrt{s}} \int_{-\infty}^{\infty} x(t)\bar{\phi}\left(\frac{t-u}{s}\right) dt$$

where $x(t)$ is a continuous signal, u is a shift parameter, s is a scale parameter, and $\bar{\phi}(t)$ is a mother wavelet [1]. A commonly used wavelet is called the Morlet wavelet, which composed of a complex exponential multiplied by a Gaussian window. The main advantage of the wavelet over the spectrogram is its ability to capture different frequencies within each window because the analysis window can not only be translated like in STFT, but it can also be dilated or contracted based on the area in question.

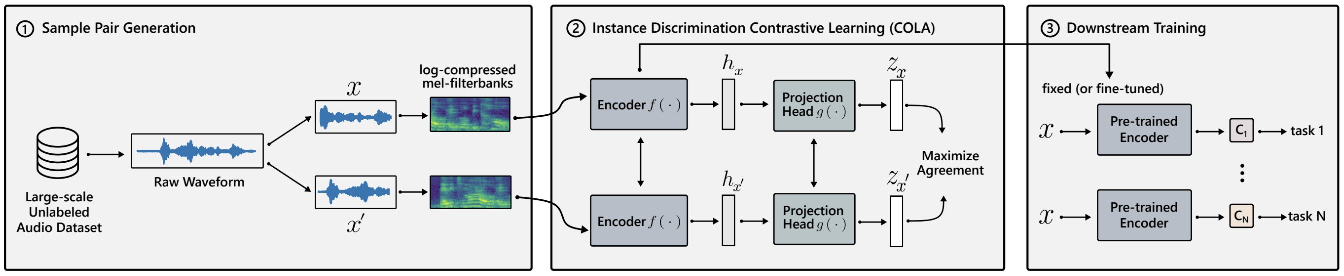


Figure 1.14: COLA training pipeline. Image Source: [27]

1.4 Contrastive Learning in Audio

There has recently been more work done using self-supervised learning in an audio setting. A majority of the work [21] uses a contrastive learning framework, a STFT based input transform, and an ImageNet-based encoder.

COLA

Saeed et al. [27] was one of the first people to incorporate contrastive learning techniques from vision into audio tasks. As shown in Figure 1.14, the authors first took audio segments and converted them into log-mel spectrograms. They then passed them into an encoder where they used a multi-class cross entropy loss, or N-pair loss. The authors then used the learned feature representations on various downstream classification tasks in environmental sounds, music, and speech recognition. They first evaluated the performance of a linear classifier trained on top of frozen representations, and then evaluated the performance of fine-tuning the encoder weights. The authors reported accuracies on the downstream tasks that outperformed the current state-of-the art self-supervised models.

CLAR

Al-Tahan et al. [33] built upon the work of SimClr [4] to expand into audio representations. They preprocessed the audio as a log Mel-Spectrogram and introduced a few more augmentations. They incorporated fading in and out in addition to the standard time and frequency masking, pitch shifting, noise injection, and time-stretching. Similar to COLA [27], they showed improved results in downstream classification tasks to evaluate their learned evaluations. They also improved upon how quickly a network could learn quality representations relative to previous methods.

Chapter 2

Experiments

2.1 Experimental Setup

I build upon the work of COLA [27] and CLAR [33] and follow a similar pipeline of work, as shown in Figure 2.1. I first go into detail about the datasets, pre-processing techniques, and encoders used.

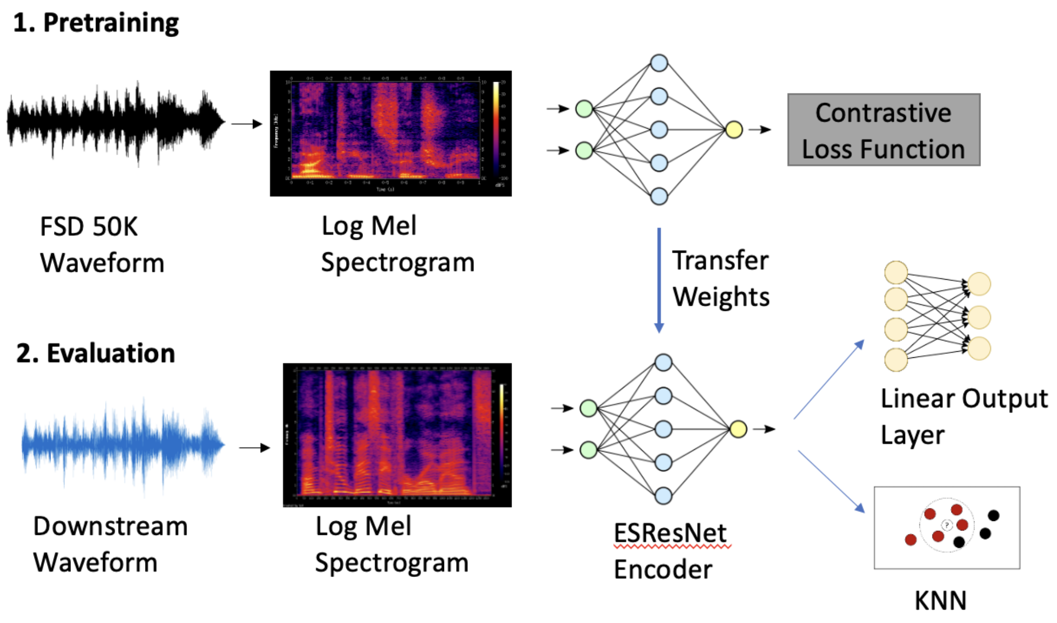


Figure 2.1: Experimental Pipeline. The experiments are run in two stages, pretraining and evaluation. A network learns a representation during pretraining and then uses that representation in a downstream classification task during evaluation.

Datasets

Pretraining Datasets

To generate the feature representations, I use FSD 50K (Freesound Dataset) [9]. This dataset contains around 50,000 human-labeled sounds encompassing roughly 200 classes that include a wide domain of human sounds, animal sounds, music, natural sounds. The audio is labeled with the AudioSet [11] hierarchical ontology, so a sound file can correspond to multiple labels. I use a smaller size dataset to provide an archetype of how contrastive learning would work in audio. In theory, a larger dataset should scale because there is a lot less of a concept of overfitting in contrastive learning.

Environmental Sounds Datasets

I evaluate the performance of the learned representations on a few datasets in the environmental sound domain. The first dataset used is ESC-50 (Environmental Sounds) [26]. This dataset contains 2000 samples divided into 50 balanced classes, with each class containing 40 5-second samples. There are 5 different overarching categories: animal, natural, human, interior, and exterior sounds. The dataset is divided into 5 pre-made folds for evaluation.

The second dataset used is called US8K (Urban Sounds) [28] and consists of 8732 audio files unequally divided into 10 classes. The 10 classes are “air conditioner”, “car horn”, “children playing”, “dog bark”, “drilling”, “engine idling”, “gun shot”, “jackhammer”, “siren”, and “street music”. There is also no uniform signal length or sample rate throughout. For evaluation, the dataset was pre-divided into 10 different folds.

The third dataset used is from the DCASE 2013 challenge [32]. This is a smaller dataset that consists of 200 total samples divided into 10 balanced classes of background noises that are ”busystreet”, ”office”, ”openairmarket”, ”park”, ”quietstreet”, ”restaurant”, ”supermarket”, ”tube”, and ”tubestation”. The dataset is divided into 2 different folds.

Music Dataset

I also evaluate the performance of the learned representations on a slightly separate task of music genre classification. I use the GTZAN [34] dataset, which consists of 1000 total music recordings divided into 10 balanced classes of 100 30s recordings. The music genres are ”blues”, ”classical”, ”country”, ”disco”, ”hiphop”, ”jazz”, ”metal”, ”pop”, ”reggae”, and ”rock. There is no official split used, so we divide the audio into a 70/15/15 Train/Valid/Test split.

Data Preprocessing and Augmentations

The input to the encoder is an audio waveform. I take signal and then augment it through random flipping, padding, cropping, and pitch shifting. I then convert the augmented signal

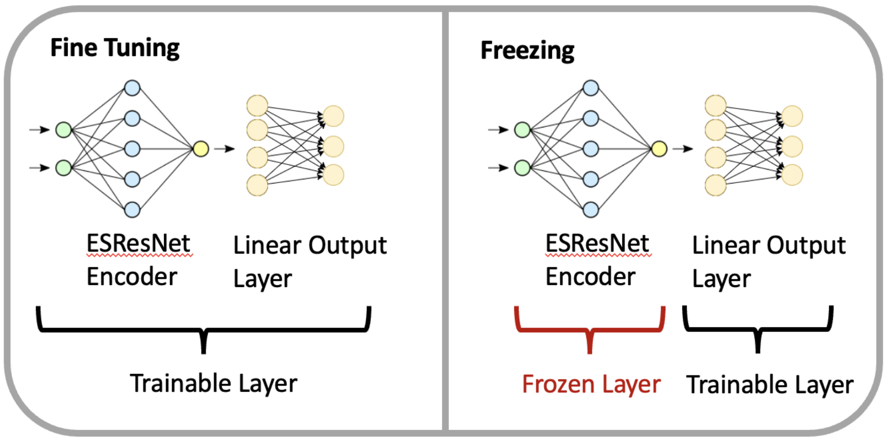


Figure 2.2: Fine-Tuning vs Freezing Experiments

into a log-Mel spectrogram. I augment the signal some more on the spectrogram with a frequency and time mask from Park et al. [25] before inputting it into the encoder.

Encoder

I follow the architecture of ESResNet [14] with some slight modifications. The authors include input channel transformations where they divide up a spectrogram into three different partitions based on frequency and stack the three partitions as separate input channels. They claim that this partitioning gives the network more information in the input, but I feel like this transformation is a bit odd. Convolutions generally highlight similar areas of interest. In their case, the kernel is passing over three relatively independent frequency ranges, seemingly making less use of the shared parameters in a convolution. Instead, for simplicity, I just stack the three log-Mel spectrograms for the three input channels. The advantage of having three input channels is that weights from traditional image-based networks can be transferred over easier. The base ESResNet model follows the vanilla ResNet50 [15] architecture.

2.2 Generating Feature Representations

I use two different contrastive learning techniques, MoCo [17] and NPID [36], to generate feature representations. Even though I am not training on ImageNet like the models were originally intended to, I reused most of the base parameters in my analysis due to time constraints. Overall results may be improved if hyper-parameters were tuned for each individual model and dataset. I use the above ESResNet as an encoder and train on FSD 50K.

2.3 Evaluating Feature Representations

I expand on the training experiments from Hornauer et al. [18] I run experiments under 3 main umbrellas: Fine Tuning pretrained weights, Freezing Pretrained weights, and KNN on top of pretrained weights. Because I am trying to show the effect of the pretraining, I only train up to 30 epochs and not until convergence. Results could be more sharp if each network was trained for longer periods of time. The output of the encoder is a single vector of length 2048, so for the fine-tuning and freezing downstream tasks, I add a fully-connected layer with an output size of the number of classes in the evaluation dataset. In the fine-tuning experiments, we allow fine-tune the encoder as well as training a classifier, but in the freezing experiments, we only train the classifier, as shown in Figure 2.2. The KNN experiments and the freezing experiments should have the exact same encoder, differing only in the final classifier. Because both NPID and MoCo make use of a memory-bank, I first repopulate the memory bank with the training features calculated from the encoder and then compute the nearest neighbors in the KNN experiments. This keeps the training and test features consistent because both features are computed and compared in the same space. The K-values in the KNN experiments were chosen through cross validation.

Chapter 3

Results

In this section, I show how contrastive pretraining with MoCo and NPID vastly improves a network's performance on downstream classification tasks. Overall, the representations from MoCo seem to translate into slightly better results than those from NPID. The specific accuracy gain from pretraining depends on both the size and diversity of each dataset. I also include visualizations about where a network might be confused.

3.1 Feature Representation Generation

As shown in Figure 3.1 and 3.2, the training loss follows a standard shape and starts to converge around epoch 120. In this type of self-supervised learning where the objective is to project features onto a lower-dimensional hyper-sphere, there is a lot less of a concept of overfitting. This is because the task is to maximize distance between different instance instead of fitting to specific training labels.

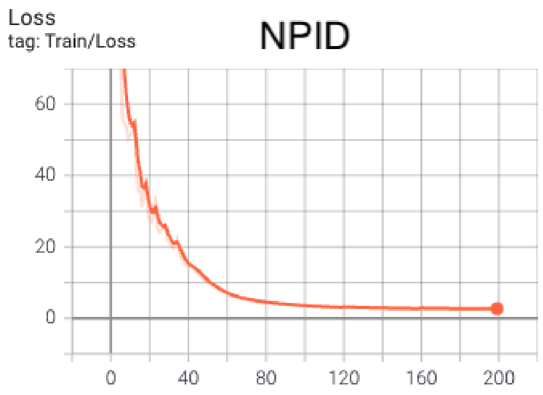


Figure 3.1: NPID Loss

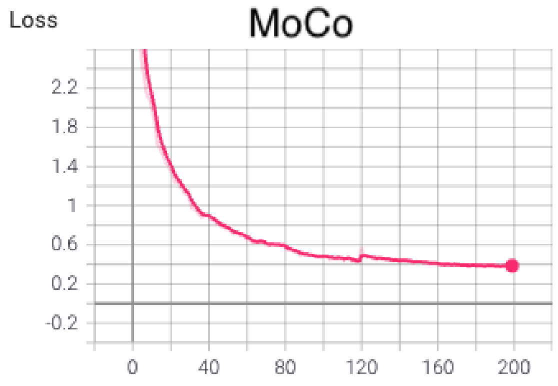


Figure 3.2: MoCo Loss

3.2 Classification

Below, I show the validation curves for the downstream datasets. Because the networks are trained with cross-validation on various folds, I display the average and bounds within 1 standard deviation of the various training curves for that epoch. I also show the best average validation accuracies in the three environmental sounds datasets that have folds and show the test set accuracy for GTZAN, which does not have folds. I include one deep dive into the outputs of the network on US8K fold 1 to understand what the different networks are seeing.

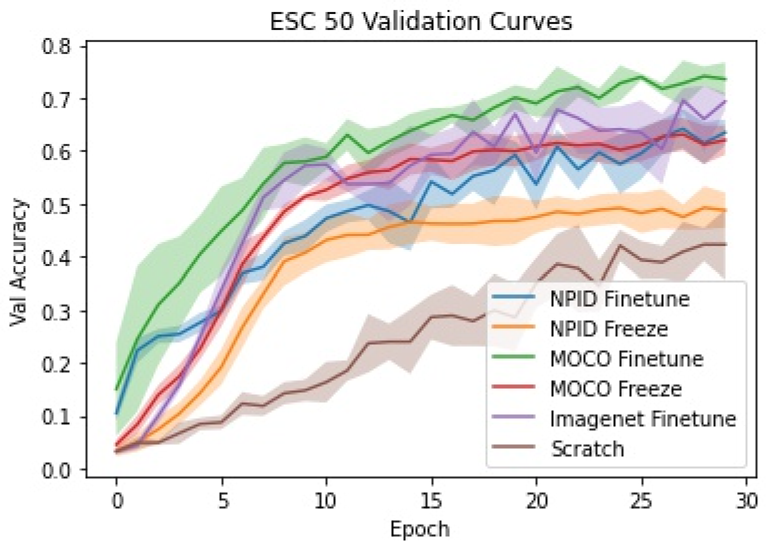


Figure 3.3: ESC-50 Validation Curves

MoCo Fine Tune	MoCo Freeze	MoCo KNN	Imagenet Fine Tune
0.742	0.595	0.468	0.696
NPID Fine Tune	NPID Freeze	NPID KNN	Scratch
0.642	0.493	0.369	0.423

Table 3.1: Best Average Validation Accuracies ESC 50

ESC-50

As shown in Figure 3.3, the initial accuracy is highest in the fine-tuned contrastive-learning pretrained networks relative to the ImageNet pretrained ones. After a few epochs, the validation accuracy of the ImageNet fine-tuned network surpasses that of the NPID pre-trained network, but not the MoCo network, which consistently shows the best performance throughout. Each one of the MoCo classifiers (fine tuning, freezing, and KNN) seems to do slightly better than the corresponding NPID one does. The network trained from scratch is also by far the slowest to converge, showing how even the frozen weights from the contrastive learning pretraining can provide a significant boost in performance. One final thing to note from Table 3.1 is that with many different classes, the KNN (K = 35) classifier performs significantly worse than the fully connected layer in the frozen experiments. This is probably because more classes introduces more noise, making clusters harder to distinguish.

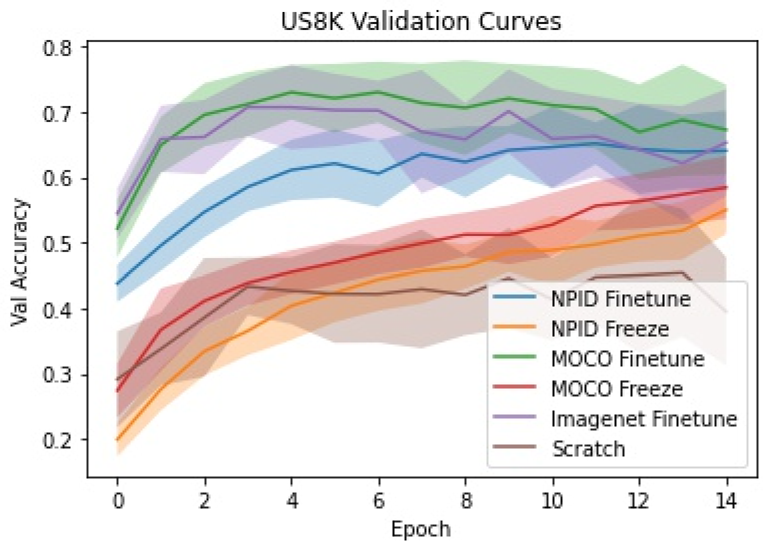


Figure 3.4: US8K Validation Curves

MoCo Fine Tune	MoCo Freeze	MoCo KNN	Imagenet Fine Tune
0.730	0.631	0.589	0.707
NPID Fine Tune	NPID Freeze	NPID KNN	Scratch
0.652	0.550	0.566	0.454

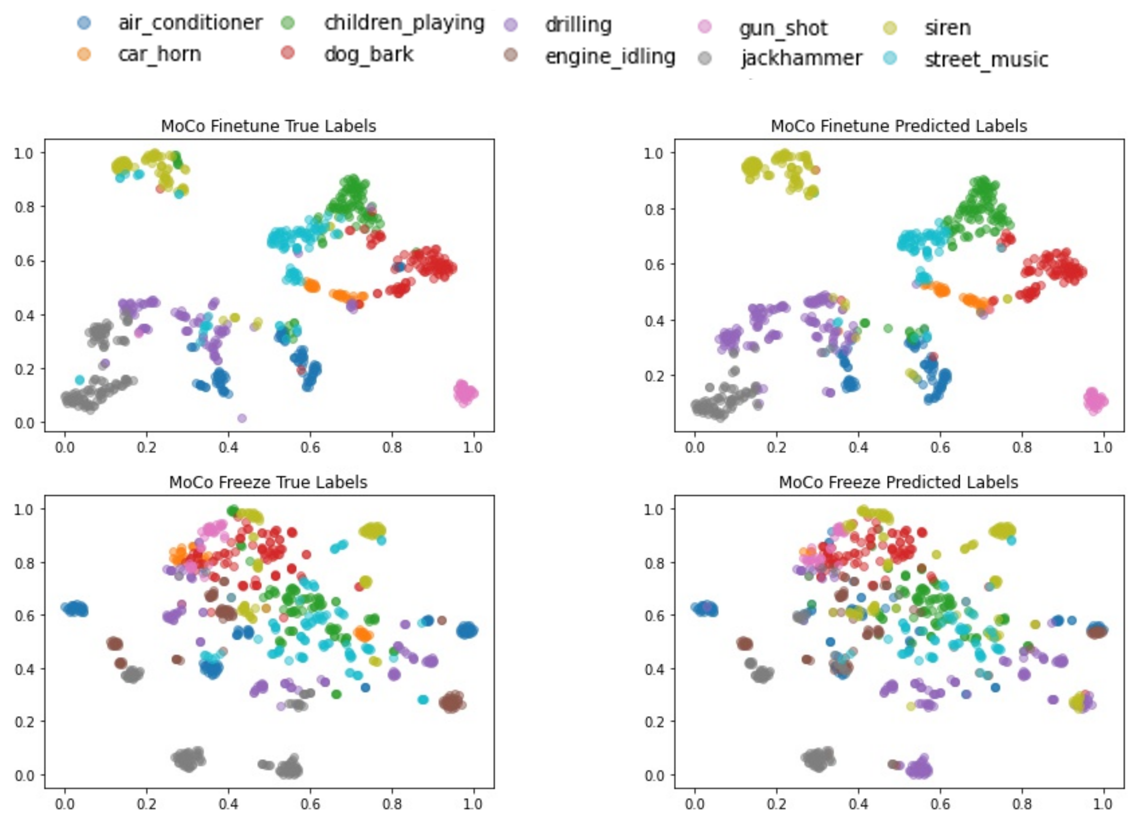
Table 3.2: Best Average Validation Accuracies US8K

US8K

As shown in Figure 3.4, the initial accuracy is highest in the fine-tuned MoCo network, with the ImageNet network following closely behind. The NPID pretrained network does not do as well relative to how well it did in ESC 50, and this might be because the US8K dataset is a lot larger. The larger dataset allows for more fine-tuning per batch, and the ImageNet pretrained network should have more variety in its features in the long run because it was trained on a larger dataset. However, the MoCo fine-tuned network still slightly outperforms the ImageNet pretrained one even in the long run. Another slight difference from the results in ESC50 is that the scratch network performs similarly compared to the frozen ones in the initial epochs. This can also be attributed to having more iterations per epoch to learn better representations faster. As shown in Table 3.2, the KNN (K = 200) and frozen best accuracies are closer than they are in ESC 50, possibly because of the fewer amount of classes.

Deeper Dive into US8K Results

I take the results of training each network on the first fold to get a glimpse of what the network is understanding. From the F1 Scores in Table 3.3, we see that drilling and engine idling are the worst classified classes in the dataset. This is corroborated by the Heat maps in Figure 3.6 and the TSNE [22] charts in Figure 3.5, showing how drilling is often confused with engine-idling and jackhammer. This makes some sense because both are longer and louder sounds that can potentially sound similar, especially as a jackhammer is often used to drill. We also see that gunshot is the best classified class overall. A gunshot is a singular loud event that lasts a short time, which is vastly different from the rest of the other classes. The TSNE graphs also show how the more "background" noises like air-conditioner and street music are clustered closer. Another slight oddity is how the clustering techniques do not predict car horn often, as both the MoCo and NPID KNN classifiers have high precisions and very low-recall for that class. This may signify how the car horn's representation does not distinguish itself well from other class's representation, confusing a KNN clustering algorithm. The TSNE graphs also display how the fine-tuned networks have clusters that are more spread apart, while the frozen encoder networks do not cluster as well. This reveals a simple visualization of the improvements from fine-tuning the encoder to just the base frozen results from the pretraining.



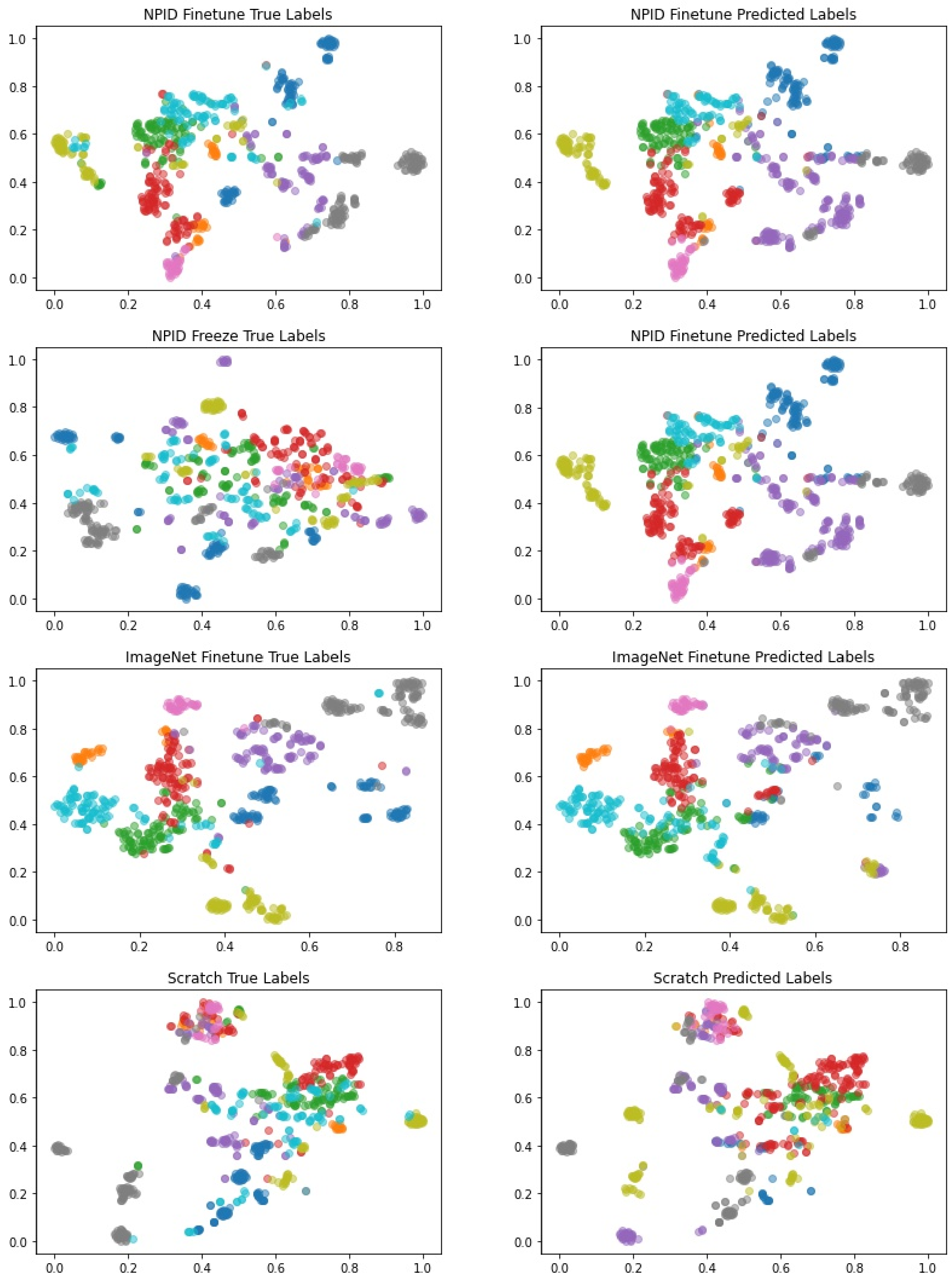


Figure 3.5: TSNEs for True and Predicted Labels for each type of network on US8K Fold 1

Class Dictionary

0: air conditioner	1: car horn	2: children playing	3: dog bark	4: drilling
5: engine idling	6: gun shot	7: jackhammer	8: siren	9: street music

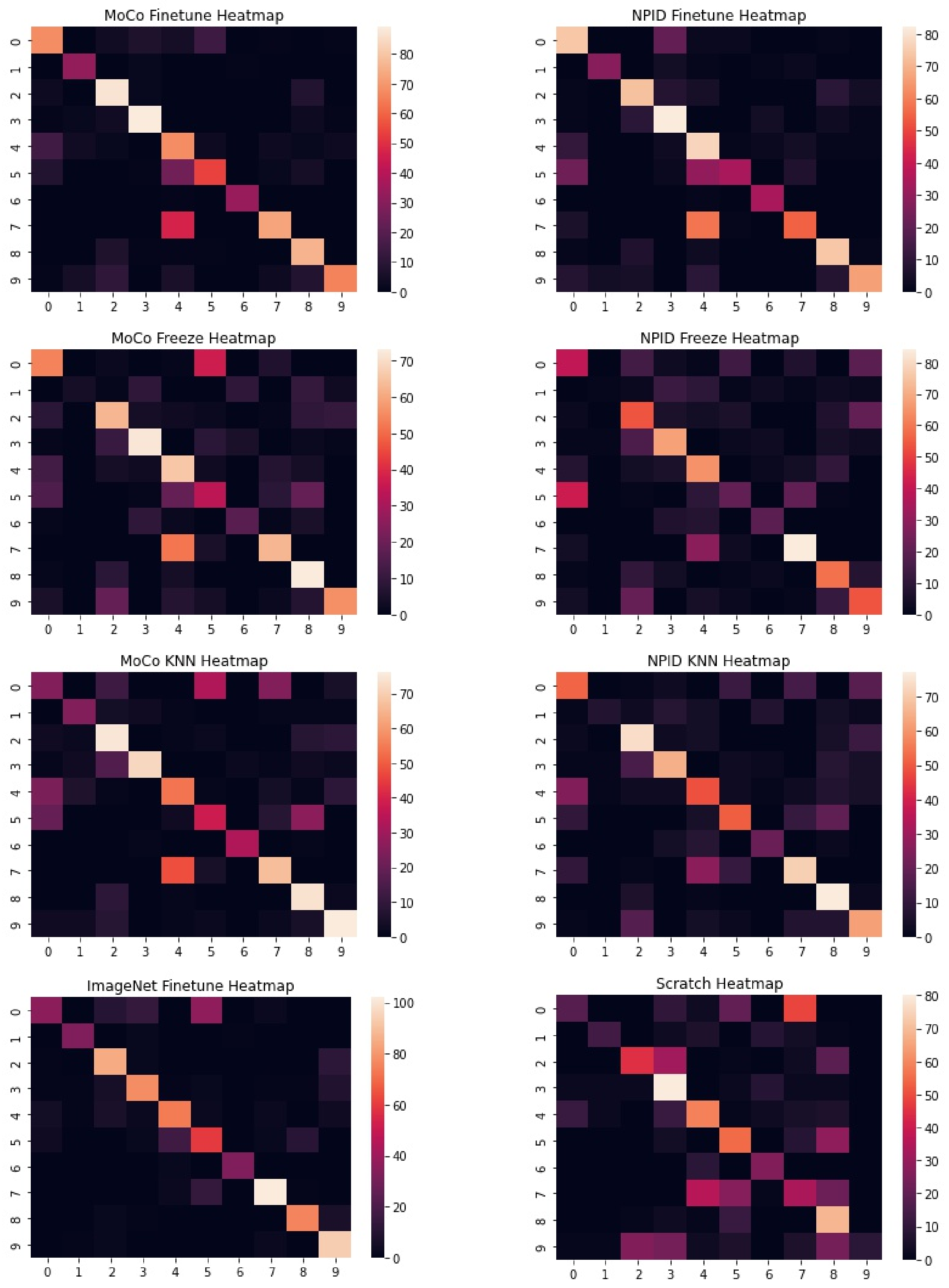


Figure 3.6: Heatmaps for Each Type of Network on US8K Fold 1

Air Conditioner

	Precision	Recall	F1 Score
MoCo Fine Tune	0.708	0.680	0.694
MoCo Freeze	0.551	0.540	0.545
MoCo KNN	0.329	0.250	0.284
NPID Fine Tune	0.592	0.740	0.658
NPID Freeze	0.391	0.400	0.396
NPID KNN	0.509	0.520	0.515
Imagenet	0.766	0.360	0.490
Scratch	0.545	0.180	0.271

Car Horn

	Precision	Recall	F1 Score
MoCo Fine Tune	0.750	0.917	0.825
MoCo Freeze	1.000	0.111	0.200
MoCo KNN	0.641	0.694	0.667
NPID Fine Tune	0.875	0.778	0.824
NPID Freeze	0.500	0.028	0.052
NPID KNN	0.778	0.194	0.311
Imagenet	0.917	0.917	0.917
Scratch	0.608	0.450	0.517

Children Playing

	Precision	Recall	F1 Score
MoCo Fine Tune	0.756	0.870	0.809
MoCo Freeze	0.583	0.630	0.606
MoCo KNN	0.600	0.750	0.667
NPID Fine Tune	0.777	0.730	0.753
NPID Freeze	0.429	0.540	0.478
NPID KNN	0.617	0.740	0.673
Imagenet	0.759	0.850	0.802
Scratch	0.608	0.450	0.517

Dog Bark

	Precision	Recall	F1 Score
MoCo Fine Tune	0.873	0.890	0.881
MoCo Freeze	0.735	0.720	0.727
MoCo KNN	0.947	0.720	0.818
NPID Fine Tune	0.683	0.820	0.745
NPID Freeze	0.636	0.680	0.657
NPID KNN	0.756	0.650	0.699
Imagenet	0.765	0.780	0.772
Scratch	0.476	0.800	0.597

Drilling

	Precision	Recall	F1 Score
MoCo Fine Tune	0.442	0.680	0.535
MoCo Freeze	0.429	0.660	0.519
MoCo KNN	0.495	0.540	0.517
NPID Fine Tune	0.414	0.770	0.538
NPID Freeze	0.492	0.650	0.560
NPID KNN	0.480	0.490	0.485
Imagenet	0.763	0.740	0.751
Scratch	0.504	0.590	0.544

Engine Idling

	Precision	Recall	F1 Score
MoCo Fine Tune	0.750	0.563	0.643
MoCo Freeze	0.366	0.354	0.360
MoCo KNN	0.469	0.396	0.429
NPID Fine Tune	0.895	0.354	0.507
NPID Freeze	0.420	0.219	0.288
NPID KNN	0.622	0.531	0.573
Imagenet	0.517	0.625	0.566
Scratch	0.444	0.573	0.500

Gunshot

	Precision	Recall	F1 Score
MoCo Fine Tune	0.971	0.971	0.971
MoCo Freeze	0.548	0.486	0.515
MoCo KNN	0.943	0.943	0.943
NPID Fine Tune	0.829	0.971	0.895
NPID Freeze	0.667	0.571	0.615
NPID KNN	0.677	0.600	0.636
Imagenet	0.971	0.943	0.957
Scratch	0.565	0.743	0.642

Jackhammer

	Precision	Recall	F1 Score
MoCo Fine Tune	0.890	0.608	0.723
MoCo Freeze	0.716	0.525	0.606
MoCo KNN	0.620	0.558	0.588
NPID Fine Tune	0.808	0.458	0.585
NPID Freeze	0.712	0.700	0.706
NPID KNN	0.670	0.592	0.628
Imagenet	0.895	0.850	0.872
Scratch	0.306	0.283	0.294

Siren

	Precision	Recall	F1 Score
MoCo Fine Tune	0.745	0.884	0.809
MoCo Freeze	0.566	0.849	0.679
MoCo KNN	0.617	0.860	0.718
NPID Fine Tune	0.771	0.860	0.813
NPID Freeze	0.602	0.686	0.641
NPID KNN	0.597	0.895	0.716
Imagenet	0.854	0.884	0.869
Scratch	0.401	0.802	0.535

Street Music

	Precision	Recall	F1 Score
MoCo Fine Tune	0.930	0.660	0.772
MoCo Freeze	0.800	0.560	0.659
MoCo KNN	0.731	0.760	0.745
NPID Fine Tune	0.904	0.660	0.763
NPID Freeze	0.500	0.540	0.519
NPID KNN	0.584	0.620	0.602
Imagenet	0.746	0.940	0.832
Scratch	1.000	0.090	0.165

Table 3.3: Class-wise F1 Scores of US8K Fold 1 for different classification methods

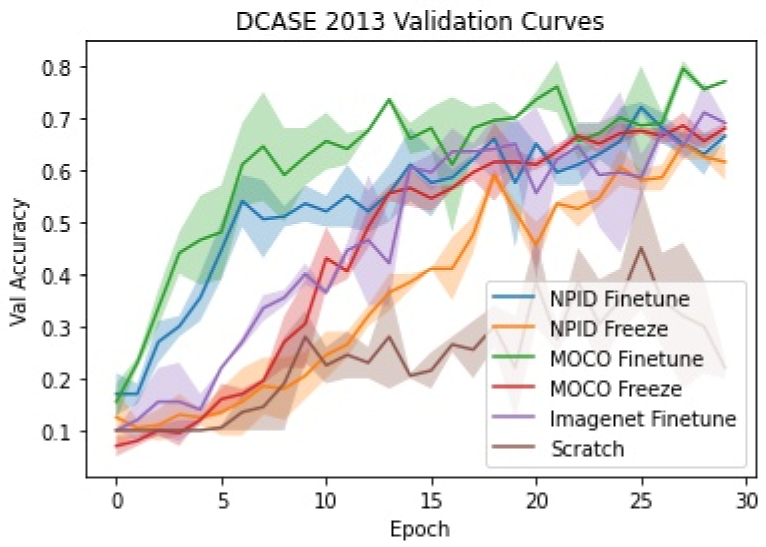


Figure 3.7: DCASE 2013 Validation Curves

MoCo Fine Tune	MoCo Freeze	MoCo KNN	Imagenet
0.795	0.685	0.830	0.710

NPID Fine Tune	NPID Freeze	NPID KNN	Scratch
0.720	0.650	0.725	0.450

Table 3.4: Best Average Validation Accuracies DCASE13

DCASE 2013

As shown in Figure 3.7, the initial accuracies for about 15 epochs are highest in the finetuned MoCo and NPID networks. The boost from the contrastive pretraining is larger than the boost in ESC 50 probably because DCASE 2013 is a significantly smaller dataset. Thus, the pretraining in a similar domain would have a higher initial boost than an ImageNet pretrained network that was trained on a significantly larger and more diverse set of data. As shown in Table 3.4, the KNN (K = 18) classifier vastly outperforms the other methods in terms of best accuracy. This is probably because with less total data, there is less noise, making it easier to cluster like data.

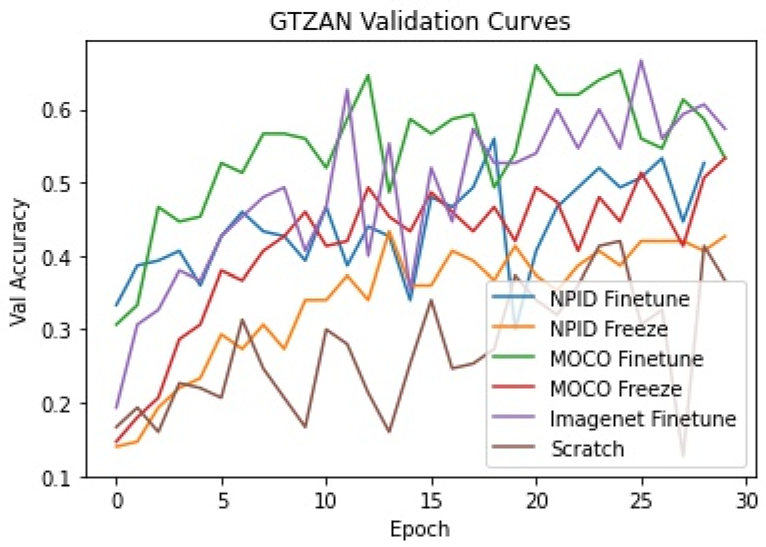


Figure 3.8: GTZAN Validation Curves

MoCo Fine Tune	MoCo Freeze	MoCo KNN	Imagenet
0.575	0.538	0.513	0.625
NPID Fine Tune	NPID Freeze	NPID KNN	Scratch
0.513	0.425	0.473	0.369

Table 3.5: Test Accuracies GTZAN

GTZAN

I investigate how the results would change in a slightly different domain of music-genre classification. As shown in Figure 3.8, the initial validation curves mirror those of ESC 50, where the MoCo and NPID pretrained networks have a slight leg up on ImageNet in the early epochs. The ImageNet pretrained network performs the best on the test set overall, which signifies that in a slightly different task, the more diverse set of features does prove advantageous relative to one that is pretrained on a smaller set of data. As shown in Table 3.5, the KNN (K = 35) classifier performs right around the same as the frozen encoder, showing little differentiation between the linear and KNN classifier for GTZAN on the new dataset.

Chapter 4

Conclusions

Overall, the experiments done have shown that contrastive-pretraining can generate features that can be used for audio classification. Even though the ResNet trained on ImageNet learns from a significantly larger amount of data than one pretrained on a smaller audio dataset, I show that MoCo and even NPID can improve early epoch training and can surpass the capabilities of the ImageNet pretrained networks in some tasks. I also show how KNN can be used on the features of the final layer to great results, especially on smaller datasets.

Future work can start with more hyperparameter tuning for these audio-specific tasks. I can also try to improve the input into the network because currently there is redundancy with inputting the same mel-spectrogram for each channel. As shown in Arias-Vergara et al. [1], there is potential in combining different types of audio transformations per channel. I also could expand the testing into larger datasets in the music domain such as the Free Music Archive (FMA) dataset [7], or I can expand the testing into different domains within audio classification. The applications of the representations can also be tested in completely different tasks such as audio detection and localization.

Bibliography

[1] T. Arias-Vergara et al. “Multi-channel spectrograms for speech processing applications using deep learning methods”. In: *Pattern Analysis and Applications* (2020).

[2] Mathilde Caron et al. “Deep Clustering for Unsupervised Learning of Visual Features”. In: *ECCV* (2018).

[3] Mathilde Caron et al. “Unsupervised Learning of Visual Features by Contrasting Cluster Assignments”. In: *NeurIPS* (2020).

[4] Ting Chen et al. “A Simple Framework for Contrastive Learning of Visual Representations”. In: *ICML* (2020).

[5] Ting Chen et al. “Big Self-Supervised Models are Strong Semi-Supervised Learners”. In: *NeurIPS* (2020).

[6] Sumit Chopra, Raia Hadsell, and Yann LeCun. “Learning a similarity metric discriminatively, with application to face verification”. In: *CVPR* (2005).

[7] Michaël Defferrard et al. “FMA: A Dataset for Music Analysis”. In: *ISMIR* (2017).

[8] Jia Deng et al. “Imagenet: A large-scale hierarchical image database”. In: *CVPR* (2009).

[9] Eduardo Fonseca et al. “FSD50K: an Open Dataset of Human-Labeled Sound Events”. In: *arXiv* (2020).

[10] Eduardo Fonseca et al. “Unsupervised Contrastive Learning of Sound Event Representations”. In: *ICASSP* (2021).

[11] Jort F. Gemmeke et al. “Audio Set: An ontology and human-labeled dataset for audio events”. In: *ICASSP* (2017).

[12] Spyros Gidaris, Praveer Singh, and Nikos Komodakis. “Unsupervised Representation Learning by Predicting Image Rotations”. In: *ICLR* (2018).

[13] Michael Gutmann and Aapo Hyvärinen. “Noise-contrastive estimation: A new estimation principle for unnormalized statistical models”. In: *AISTATS* (2010).

[14] Andrey Guzhov et al. “ESResNet: Environmental Sound Classification Based on Visual Domain Models”. In: *arXiv* (2020).

[15] Kaiming He et al. “Deep Residual Learning for Image Recognition”. In: *CVPR* (2016).

[16] Kaiming He et al. “Improved Baselines with Momentum Contrastive Learning”. In: *CVPR* (2020).

[17] Kaiming He et al. “Momentum Contrast for Unsupervised Visual Representation Learning”. In: *CVPR* (2020).

[18] Sascha Hornauer et al. “Unsupervised Discriminative Learning of Sounds for Audio Event Classification”. In: *ICASSP* (2021).

[19] Ying Hu and Guizhong Liu. “Dynamic characteristics of musical note for musical instrument classification”. In: *IEEE* (2011).

[20] Alexander Kolesnikov, Xiaohua Zhai, and Lucas Beyer. “Revisiting Self-Supervised Visual Representation Learning”. In: *CVPR* (2019).

[21] Shuo Liu et al. “Audio Self-supervised Learning: A Survey”. In: *arXiv* (2022).

[22] Laurens van der Maaten and Geoffrey Hinton. “Visualizing Data using t-SNEd”. In: *Journal of Machine Learning Research* (2008).

[23] Mehdi Noroozi and Paolo Favaro. “Unsupervised Learning of Visual Representations by Solving Jigsaw Puzzles”. In: *ECCV* (2016).

[24] Aaron van den Oord, Yazhe Li, and Oriol Vinyals. “Representation Learning with Contrastive Predictive Coding”. In: *arXiv preprint arXiv:1807.03748* (2018).

[25] Daniel S. Park et al. “SpecAugment: A Simple Data Augmentation Method for Automatic Speech Recognition”. In: *Proc Interspeech* (2019).

[26] Karol J. Piczak. “ESC: Dataset for Environmental Sound Classification”. In: *ACM MM* (2015).

[27] Aaqib Saeed, David Grangier, and Neil Zeghidour. “Contrastive Learning of General-Purpose Audio Representations”. In: *arXiv* (2020).

[28] Justin Salamon, Christopher Jacoby, and Juan Pablo Bello. “A Dataset and Taxonomy for Urban Sound Research”. In: *ACM MM* (2014).

[29] Florian Schroff, Dmitry Kalenichenko, and James Philbin. “FaceNet: A Unified Embedding for Face Recognition and Clustering”. In: *CVPR* (2015).

[30] Kihyuk Sohn. “Improved Deep Metric Learning with Multi-class N-pair Loss Objective”. In: *NeurIPS* (2016).

[31] Hyun Oh Song et al. “Deep Metric Learning via Lifted Structured Feature Embedding”. In: *CVPR* (2016).

[32] Dan Stowell et al. “Detection and classification of acoustic scenes and events”. In: *IEEE Transactions on Multimedia* (2015).

[33] Haider Al-Tahan and Yalda Mohsenzadeh. “CLAR: Contrastive Learning of Auditory Representations”. In: *arXiv* (2020).

[34] George Tzanetakis, Georg Essl, and Perry Cook. “Automatic Musical Genre Classification Of Audio Signals”. In: *ISMIR* (2001).

[35] Xudong Wang, Ziwei Liu, and Stella X. Yu. “Unsupervised Feature Learning by Cross-Level Instance-Group Discrimination”. In: *CVPR* (2021).

[36] Zhirong Wu et al. “Unsupervised Feature Learning via Non-Parametric Instance Discrimination”. In: *CVPR* (2018).

[37] Richard Zhang, Phillip Isola, and Alexei A. Efros. “Colorful Image Colorization”. In: *ECCV* (2016).