Delft University of Technology
Master of Science Thesis in Embedded Systems

Unsupervised Facial Expression Recognition using Periocular Images

Yuan Fu

Embedded Networked Systems

TU Delft Delft University of Technology

Unsupervised Facial Expression Recognition using Periocular Images

Master of Science Thesis in Embedded Systems

Embedded and Networked Systems Group
Faculty of Electrical Engineering, Mathematics and Computer Science
Delft University of Technology
Mekelweg 4, 2628 CD Delft, The Netherlands

Yuan Fu
Y.Fu-6@student.tudelft.nl

10/10/2022

Author

Title
Unsupervised Facial Expression Recognition using Periocular Images
MSc Presentation Date

17/10/2022

Graduation Committee

Marco Zuniga	Delft University of Technology
Guohao Lan	Delft University of Technology
Jie Yang	Delft University of Technology

Abstract

Facial expression recognition on head-mounted devices (HMDs) is an intriguing research field because of its potential in various applications, such as interactive virtual reality video meetings. Existing work focuses on building a supervised learning pipeline that utilizes a vast amount of labeled periocular images taken by the built-in cameras. However, the labeling process requires intensive manual work, which is costly and time-consuming. In this thesis, we apply self-supervised learning techniques that leverage unlabeled periocular images to learn representations for facial expression recognition tasks. When the model is trained through self-supervised learning, it is transferred as a feature extractor to form multiple individual inference models. In addition, as most deep-learning models only accept low-resolution images due to heavy computing overload with large input sizes, images taken by modern cameras on HMDs are often aggressively down-sampled to match the model input. To improve the model's usability and speed performance, we train the models using the frequency representations of images rather than conventional RGB pixel arrays. We conducted extensive evaluation experiments with a dataset of 23 invited volunteers. The results show that the self-supervised models achieve comparable performance with existing approaches, and, at the same time, the number of labels used is reduced by approximately 99%. By training in the frequency domain, the model is 1.89 times faster in training and 1.23 times faster in inference with reduced input size.

Preface

This thesis presents my work on the graduate project for the Master of Science in Embedded Systems at TU Delft. Through this journey, I gained abundant knowledge and experience regarding facial expression recognition, deep learning, and computer vision. During my research on these topics, I found great potential in deep learning on embedded devices, especially its capabilities for provoking life-changing concepts.

I would like to express my gratitude to Dr. Guohao Lan for his guidance, detailed feedback, and professional advice during the research. I would also like to thank Lingyu for his support and suggestions during the challenging periods. Additionally, I would like to thank Marco for his help and time. Besides, thank Jie for being on my thesis committee. Lastly, thank all the volunteers who participated in our data collection experiment and my families and friends who encouraged me and supported me all the time.

Yuan Fu

Delft, The Netherlands
10th October 2022

Contents

Chapter 1

Introduction

Facial expressions are an essential component of human communication. They serve as means to convey people's intentions and internal states in a non-verbal manner. Due to the increasing popularity of mobile and embedded devices, it is desirable to implement a facial expression recognition (FER) platform with head-mounted devices (HMDs), such as virtual reality (VR). By directly using periocular images taken by the built-in cameras, FER on HMDs serves as a way for more immersive and interactive applications such as VR games [76], and remote communication [38]. Empowered by deep learning, FER gains promising performance by learning from massive labeled data. However, many deep learning models have been focusing on a supervised pipeline where much effort has been put into building a dataset with clean expression labels [38, 47, 50]. The requirement for massive labels limits the adoption of deep learning models as the annotation process is expensive and labor-intensive. Hence, it is imperative to investigate alternative methods that realize FER without numerous manual annotations while maintaining comparable performance.

Among the approaches to FER, convolutional neural networks (CNNs) have shown promising performance by learning directly from input images without building hand-craft features [8, 28, 53]. However, limited by computing resources and memory capacity, most CNN models only accept low-resolution images, for example, $224 \times 224 \times 3$ for Residual Neural Network (ResNet) [37]. Modern cameras often produce images with much larger resolutions, e.g., 1024 x 1280. As a result, the input images will be down-sampled to match the input size accepted by most CNN models, which may cause information loss and performance drop. Therefore, it is essential to reduce the input size without sacrificing performance and information to train an effective and efficient CNN model. In addition, interactive scenarios on HMDs, for example, an online VR meeting, are latency-sensitive. A delay in expression recognition may cause a mismatch between the dialogue context and the non-verbal messages conveyed by the facial expressions. However, many solutions to faster FER often include extra steps of extracting discriminative features [29, 75], which usually work well only in specific conditions (e.g., fixed head pose and appropriate lighting). Hence, it is beneficial to reduce the response latency of FER without extra design and effort.

In this thesis, we will explore to answer the following research questions: a) how can the FER using periocular images be realized without substantial manual

annotations? b) how to achieve comparable performance with a reduction in the input size and the response latency?

To tackle the challenge of data annotation, we apply self-supervised representation learning that utilizes unlabeled data for FER tasks. Self-supervised learning models can learn general representations from massive periocular images without manual labeling. After the training, the models are calibrated with small-scaled labeled samples to form individual inference models. Among the self-supervised learning models, we apply contrastive learning to process the unlabeled data. Contrastive learning is a discriminative method that learns features to group similar samples and set apart dissimilar groups from each other. Unlike the generative approach that learns by building a distribution from the data, contrastive learning avoids the costly generation step and achieves state-of-art performance in classification tasks [12, 13, 36, 63]. To reduce the input size and accelerate CNN models, We train the CNN models in the frequency domain to better preserve image information instead of the conventional RGB color space. The frequency spectrum of an image is obtained by decomposing the image into different frequency channels, which have different information densities [71]. Instead of aggressively down-sampling the input in the RGB domain, we can reduce the input size by discarding frequency channels that are low in information compaction.

To evaluate the effectiveness of our proposed solutions, we conduct experiments with periocular images collected from 23 participants. We demonstrate that our approach, based on self-supervised learning, reaches a close performance with its supervised counterpart in the subject-specific FER tasks. At the same time, the usage of expression labels is reduced by approximately 99%. We further evaluate the performance of the model in terms of latency and demonstrate that when only using a subset of the frequency components as the input, the model gains a boosted acceleration in 1.89 times faster training and 1.23 times faster inference while maintaining a comparable prediction accuracy with its RGB counterpart.

In summary, the main contributions of the thesis are highlighted as follows:

- This thesis applies a discriminative self-supervised learning model (i.e., contrastive learning) that leverages massive unlabeled periocular images to extract general features for FER tasks. After learning the features, several subject-specific inference models are trained using small-scaled labeled data, which significantly alleviates the work and cost of data annotations in conventional supervised training.
- We collected a unique dataset of 23 subjects which contains periocular images with expression labels. The dataset is made public and can be used in future studies.
- We convert the learning input from the conventional RGB domain to the frequency domain and train the CNN model only using a subset of the frequency channels. The proposed method speeds up the training and inference by 1.89 and 1.23 times respectively, while maintaining a comparable accuracy to the conventional RGB-based solutions.

The remainder of the thesis is structured as follows: Chapter 2 presents the background concept of the study topic. Related work regarding FER and learning in the frequency domain is presented in Chapter 3. Chapter 4 describes the

main design of our training pipeline, methods, and algorithms. The evaluation and results are presented in Chapter 5. Finally, Chapter 6 concludes the thesis and discusses future work.

Chapter 2

Background

This Chapter presents background information on the research field. Section 2.1 provides general information regarding self-supervised learning. Section 2.2 describes essential concepts of contrastive learning. Section 2.3 briefs the process of converting images from the color space to the frequency domain (i.e., discrete cosine transform (DCT)) and the properties of DCT coefficients.

2.1 Self-supervised Learning

Currently, the mainstream machine learning methods are supervised-based, which relies much on manually annotated data. With a large annotated dataset and increasingly complex neural network structures, supervised learning is performing well in tasks such as image classification [16, 39], natural language processing [17], and object detection [56]. However, compared with unlabeled data, only a small part of data is labeled in the big data era, and it is costly to gather data with clean labels. In tackling this challenge, self-supervised learning has been developed to learn representations from unlabeled data. Self-supervised learning has shown to be promising in computer vision tasks [10, 14, 31] as it directly learns from data without hand-craft labels to explore inherent features inside data, and the learned knowledge is transferable to various types of downstream tasks.

Self-supervised learning is proceeded by solving a pretext task. A pretext task is a task we set for the model to solve in order to learn visual representations from data. Generally, the self-supervised learning process can be summarized as follows:

- Produce pseudo-labels from data itself.
- Predict or recover part of the samples based on the transformed data. This step is to build the pretext task. By solving this task, the model can progress to learn features.

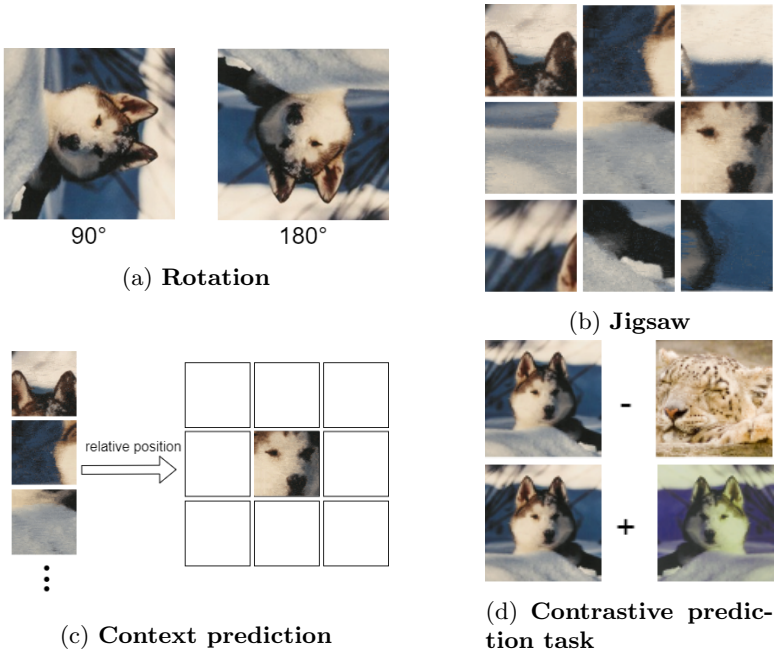
For example, a rotation is applied as the transformation function, and given the rotated and original images, an encoder is tasked to predict the rotation degrees [27]. The rotated degrees serve as the pseudo-labels generated randomly by the transformation function. The encoder needs to "recover" the image by predicting the rotation. This state-of-the-art method has proven promising in

object recognition, object detection, classification, etc. The instinct behind this is that in order to predict the rotation angle, the encoder must learn to extract object boundaries and visual representations invariant to the rotation.

Generative vs discriminative. Generally, self-supervised learning can be categorized as either generative or discriminate methods [18]. Generative methods train an encoder to encode the inputs into embedding features, from which the encoder will reconstruct the original inputs. These methods approach learning the internal data distribution with various mechanisms, including auto-encoding models [42, 68] where an encoder and a decoder are used to reconstruct inputs, auto-regressive models [65, 67], which aim to model images pixel by pixel, and hybrid models [23, 72]. Generative methods are often computationally costly because they require learning high-level details from the data. Unlike generative models, discriminative models do not have the expensive generation step and directly learn the boundaries among samples. Discriminative learning uses an encoder to extract representations that can group similar samples and spread out dissimilar ones by comparing them with each other. There are discriminative methods in a context perspective [18, 49], which aim to compare the feature from a local context with the global context, and in an instance perspective [9, 12, 62], which focuses on the relationship between instance-level representations.

Pretext Tasks. Unlike supervised learning, self-supervised learning employs a pretext task to act as a task that an encoder network needs to solve using the visual representation acquired in the training process. A well-designed pretext task can train a model to learn more general features for downstream tasks. Example pretext tasks are described as follows (visualized in Figure 2.1):

- **Predicting image rotation** [27]. The encoder needs to predict the rotation degrees from the rotated image. The encoder learns to predict the correct degrees by recognizing rotation-invariant content.
- **Solving a jigsaw puzzle** [51]. The jigsaw pretext task requires the encoder to recover a jigsaw puzzle to the original image. By solving this task, the networks can learn the spatial relations of the patches by recognizing the object.
- **Context prediction task** [18]. The encoder solves the task by predicting the relative position of a patch. Given a central patch of an image, the encoder should predict the relative positions of other patches to the central one. This task works similarly to solving a jigsaw puzzle, except context prediction is more straightforward.
- **Contrastive prediction task**. In this task, the encoder should differentiate similar/dissimilar samples. Through this task, the encoder learns representations that bring similar samples close to each other and maximize the distance between dissimilar groups. A commonly used task is **Instance discrimination**, where each instance in the dataset is regarded as a separate class. The motivation of instance discrimination is based on the observation that the top classification responses from a network trained with supervision are visually correlated, which reveals that supervised learning can capture visual similarity among classes even though they are annotated as different classes [70]. This suggests that neural networks can obtain similar class-invariant visual representations.



(a) **Rotation**

(b) **Jigsaw**

(c) **Context prediction**

(d) **Contrastive prediction task**

Figure 2.1: **Examples of 4 pretext tasks: (a) The original image is rotated around a set of degrees (90, 180, 270, degrees, etc.). (b) A Jigsaw puzzle is generated from an image, and the puzzle should be recovered to the original image. (c) An image is divided into nine patches. The task is to predict their relative position to the central patch. (d) The original and its augmented version are considered positive samples. The task is to predict similar samples or discriminate against dissimilar samples given a list of pairs.**

2.2 Contrastive Learning

Contrastive learning is a typical discriminative self-supervised learning method that achieves state-of-the-art performance. It trains the encoder to learn general features from unlabeled data by differentiating the representation vectors of similar/dissimilar samples.

Figure 2.2 illustrates an overview of the contrastive learning process. It starts by selecting a data sample as the anchor. By definition, data points in the same distribution of the anchor are called positive samples, whereas those from a different distribution are called negative samples. The samples are generated by applying image augmentation to the original samples. The main goal of augmentation is to make the model learn transformation-invariant features. During the training process, the model learns to minimize the distance between positive samples and, at the same time, maximize the distance between the anchor and its negative samples. This can be achieved using a contrastive loss function. The model is trained by minimizing the contrastive loss function and backpropagating the gradients.

The design of contrastive loss is essential to the training, and it varies based on pretext tasks. Instance discrimination is an effective and commonly used task,

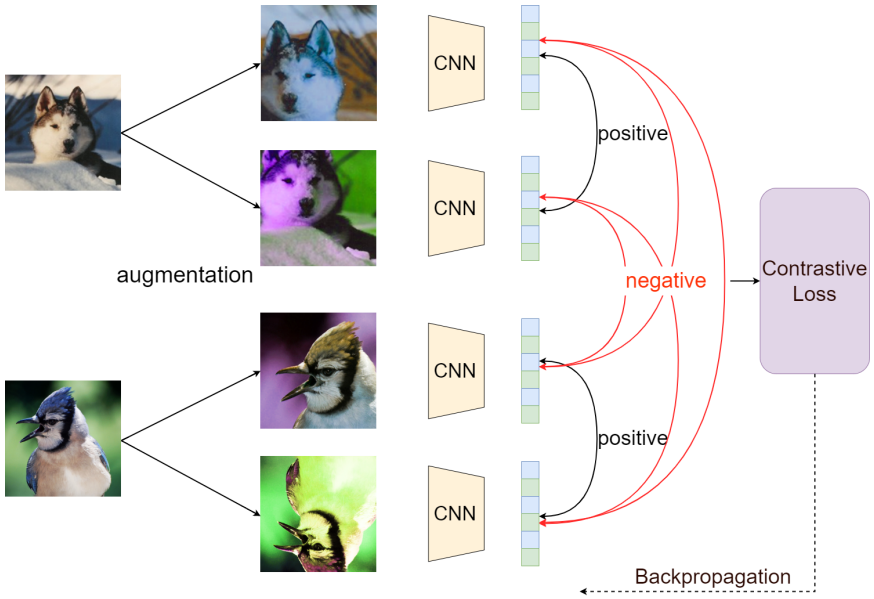


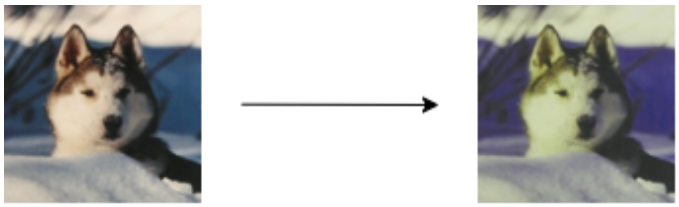
Figure 2.2: **Principal workflow of contrastive learning. Augmentation will first be applied to the anchor images to generate positive and negative samples. The samples are fed into CNNs to produce representations. A contrastive loss is obtained by contrasting the representations. The CNNs will be updated by optimizing the loss.**

where contrastive learning has been shown to achieve strong generalization in transfer learning [78]. In instance discrimination, the loss is calculated at the instance level by treating each sample as a unique class. Good results have been obtained in work such as SimCLR [12] and MoCo [36].

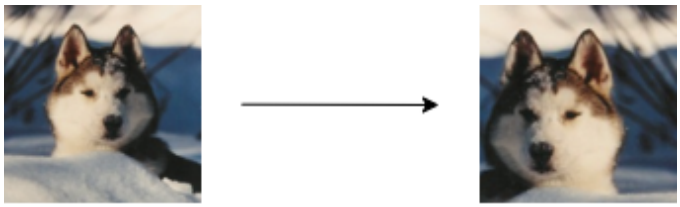
2.2.1 Image Augmentation

In self-supervised learning, image augmentation serves as a way to produce positive/negative samples by generating different views of images. Study shows that a good selection of augmentation functions is essential to learn good representations in contrastive learning [12]. In the thesis, there are two used augmentation functions: appearance and geometric transformations (visualized in Figure 2.3), which are described as follows:

- **Appearance Transformation**. Appearance transformation revolves around pixel-level color adjustments, including but not limited to Gaussian blur, noise, and color jitter (including color dropping, change of hue, brightness, contrast, and saturation). This transformation trains the networks to learn appearance-invariant features.
- **Geometric Transformation**. Geometric transformation changes the spatial alignment of pixels, including cropping and resizing, cutout, rotating, and flipping (horizontally and vertically). It makes the encoder extract invariant features from different geometric placement views.



(a) **Appearance Transformation: the original pixel-level color distribution is changed.**



(b) **Geometric Transformation: the original geometric alignment of pixels is changed.**

Figure 2.3: **Examples of image augmentation.**

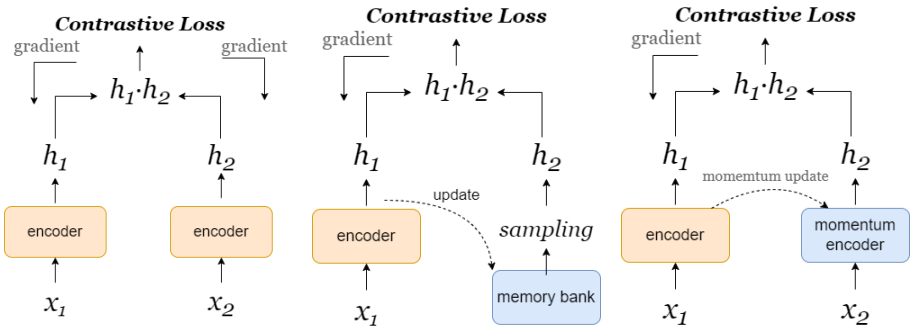
2.2.2 Contrastive Learning Architectures

The process of self-supervised learning can be seen as building a dynamic dictionary that gives good representations of the input samples for the contrastive learning goal [36]. Like a dictionary, similar keys (synonyms) are grouped closely for easy reference, while dissimilar keys (antonyms) are separated clearly. Depending on how to build the dictionary, contrastive learning can be categorized into three architectures: end-to-end, memory bank, and momentum. An overview of their comparison can be found in Figure 2.4.

End-to-end

In end-to-end learning (e.g., [52, 73]), the dictionary is comprised of encoded samples from the current batch. Two encoders are used to extract representations from the input samples. Positive pairs are those from the same distribution (e.g., two augmented views of the same image). Negative samples are from different data distributions (e.g., any augmented views of images from other classes). The contrastive loss is calculated based on representations computed by the two encoders, and the weights in the encoders are updated end-to-end through backpropagation.

Even though end-to-end is the most natural and direct way to train encoders, this architecture has one drawback: the dictionary size is coupled with the training batch size. A larger dictionary contains more representations of negative samples, leading the encoder to a potentially better performance by learning from more negative samples. The ideal situation for the end-to-end method would be to use the whole dataset as a batch. However, limited by GPU memory size and significant computation overhead in large batch size, the end-to-end method is always suboptimal and has a high demand for computing resources.



(a) **End-to-end** (b) **Memory Bank** (c) **Momentum**

Figure 2.4: **Conceptual comparison of three contrastive learning architectures.** x_1, x_2 **represent two input sample vectors while** h_1, h_2 **are corresponding representation vectors. (a)Two encoders encode** x_1, x_2 **into representation vectors** h_1, h_2**. The dictionary only consists of** h_2**. Each encoder is updated through the gradient of the contrastive loss. (b)A memory bank stores representations of all samples in a dataset. Each batch randomly sampled representations from the memory bank as a pool of negative samples. (c) The momentum encoder is updated with a weighted average based on the encoder updated with gradients.**

Memory Bank

Using a large batch size remains an issue in end-to-end training. A memory bank [70] provides a possible solution by separately storing the representations of all samples in the dataset instead of building a dynamic encoder for mapping the samples. Without the encoder that maps the sample to the dictionary, it becomes possible to maintain a large dictionary since there are fewer parameters to be updated using backpropagation. Unlike end-to-end, where the dictionary is updated by optimizing the mapping encoder, the memory bank is updated by replacing stored representations generated by the representation extractor.

A memory bank decouples batch size with dictionary size, allowing us to include more negative samples without increasing training batch size. However, the representations in the bank are always outdated because only the representations of the current mini-batch can be updated, and the sampled representations from a memory bank could be generated by the old representation extractor, which is inconsistent with the current representation extractor.

Momentum

One way to accommodate a large dictionary while keeping the representations consistent is using the momentum encoder [36]. It uses a queue as the dictionary in which the representations newly generated by the momentum encoder are enqueued while the earliest representations will be eliminated. By constantly updating the dictionary, the inconsistency issue of the memory bank is alleviated.

The momentum encoder is updated with momentum by applying the weighted

average of the old weights of the momentum encoder and the new weights of the representation extractor. The momentum update keeps the momentum encoder up-to-date without computing gradients, enabling support for a large dictionary.

2.2.3 Loss Function in Contrastive Learning

In this section, we review the basic concepts of contrastive loss. A contrastive loss [34] is a metric that is small in value when the anchor is similar to its positive samples and dissimilar to negative samples. The principle for designing a contrastive loss function is that given an anchor image x, its positive samples x^+ and negative samples x^-, and an encoder function $f(.)$, the similarity for positive pairs should be much greater than that with negative samples:

$$sim(f(x), f(x^+)) \gg sim(f(x), f(x^-)) \tag{2.1}$$

Firstly, we explore the similarity metric. In contrastive learning, the most commonly used similarity metric is **cosine similarity**. Given two vectors a and b, the cosine similarity is defined as:

$$cos\_sim(a, b) = \frac{a \cdot b}{\|a\|\|b\|} \tag{2.2}$$

Unlike Euclidean distance, which measures similarity based on the distance in the Euclidean space, cosine similarity measures the cosine value of the angle between two vectors regardless of the length. If the $cos\_sim(a, b)$ is 1, a and b point in the same direction and reach the maximum cosine similarity. In contrastive learning for image tasks, the data are often very high in dimension. It is more reasonable to use cosine similarity since it measures similarity in a low-dimensional space based on orientation instead of magnitude.

Cross-entropy is the most common loss function in machine learning to measure the difference between predictions and labels by capturing the difference between two probability distributions. The idea is also used in contrastive loss functions to express dissimilarity.

For a binary classification task, cross-entropy is defined:

$$L_i = -[y_i \cdot log(pi) + (1 - y_i) \cdot log(1 - p_i)], \tag{2.3}$$

where y_i is the label of samples i, and p_i stands for the prediction probability of sample i. For example, when sample i is true, and its prediction is true, then we have $L_{Xent} = 0$, which means it predicts perfectly.

It can be extended to the multi-classification problem. The multi-class cross-entropy loss for sample i is defined as:

$$L_{i,c} = -\sum_i [y_{i,c} \cdot log(p_{i,c})], \tag{2.4}$$

where $y_{i,c}$ is a binary indicator (0 or 1) whose value is 1 when sample i is correctly predicted as class c. $y_{i,c}$ denotes the probability of i being classified as c. The total loss is $\sum_{c=1}^{k} L_{i,c}$. When the class labels are denoted by one-hot encoding, the total loss can be simplified as:

$$L_{Xent} = -\sum_{i=1}^{k} log(p_i), \tag{2.5}$$

where p_i denotes the probability of sample i being correctly predicted, and k denotes the number of classes.

Cross-entropy is often combined with softmax function. The softmax for output x_+ is defined as:

$$softmax(x_+) = \frac{exp(x_+)}{\sum_{i=0}^{k} exp(x_i)}, \tag{2.6}$$

where k denotes the number of classes. Softmax converts the outputs of a dense layer (logits) into probabilities. With the probability distribution of the outputs, the cross-entropy loss for x_+ based on softmax can be computed:

$$L_+ = -log(\frac{exp(x_+)}{\sum_{i=0}^{k} exp(x_i)}) \tag{2.7}$$

Contrastive loss of two sample vectors is given by Chopra et al. [15] in a siamese architecture. Given a pair of inputs (x_i, x_j) and their labels (y_i, y_j), the loss would be minimized if their representations are from the same class and maximized if they are from different classes:

$$\ell(x_i, x_j) = \begin{cases} \|f_\theta(x_i) - f_\theta(x_j)\|_2^2, & y_i = y_j \\ max(0, (\alpha - \|f_\theta(x_i) - f_\theta(x_j)\|_2)^2), & y_i \neq y_j \end{cases} \tag{2.8}$$

where θ represents the parameters to be optimized in the encoder function $f_{\theta(.)}$, α is a hyperparameter that defines the lowest distance for a sample pair from different classes.

Based on the siamese definition of contrastive loss, Schroff et al. [58] proposed **triplet loss** that expresses contrastive loss by comparing the anchor input x with its positive samples x^+ and its negative sample x^- at the same time:

$$\ell_{triplet} = \sum_{i}^{N} max(0, \|f(x_i) - f(x_i^+)\|_2^2 - \|f(x_i) - f(x_i^-)\|_2^2 + \alpha) \tag{2.9}$$

where α is a hyperparameter that defines the lower bound of the triplet loss for triplet(x, x^+, x^-) and N is the number of triplets in the training set.

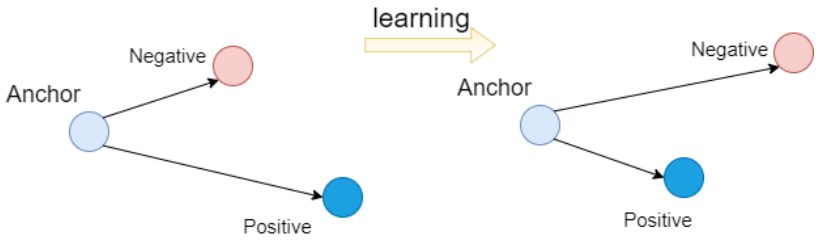


Figure 2.5: **Triplet loss is large if the anchor is close to its negative or far from its positive. By optimizing triplet loss, the anchor will be closer to the positives and farther to its negatives.**

Computing triplet loss will become impractical if the number of classes is large. Sohn et al. [60] proposed a generalized triplet loss called **Multi-Class**

N-pair loss, which incorporates multiple negative samples in the loss function. With (N+1) triplets in the training set $\{x, x^+, x_1^-, ..., x_i^-, ..., x_{N-1}^-\}$, where x^+ is the positive sample to anchor x and x_i^- are negatives, the N-pair loss is defined as:

$$\ell(x, x^+, \{x_i^-\}_{i=1}^{N-1}; f(.;\theta)) = log(1 + \sum_{i=1}^{N-1} exp(f(x)^\top f(x_i^-) - f(x)^\top f(x^+))) \tag{2.10}$$

where $f(.;\theta)$ is the encoder function whose parameters θ need to be optimized.

If we define negative samples as noise samples, we are able to reduce the computation complexity by using **Noise Contrastive Estimation** (NCE) [33] to measure contrastive loss. The idea is to train a logistic regression classifier to differentiate the target sample from noise samples. Given the target sample x in distribution $p_\theta(x) = P(x|C = 1;\theta)$, and the noise sample $\tilde{x}$ in distribution $q(\tilde{x}) = P(\tilde{x}|C = 0)$. We will have $P(\tilde{x}|C = 0) + P(x|C = 1;\theta) = 1$ by definition.

The logit in terms of parameters θ for the sample x in the target distribution is:

$$logit_\theta(x) = log\frac{p_\theta(x)}{1 - p_\theta(x)} = log\frac{p_\theta(x)}{q(x)} \tag{2.11}$$

Based on the logits, we can calculate the probability with sigmoid function $\sigma(.)$:

$$h_\theta(x) = \sigma(logit_\theta(x)) \tag{2.12}$$

By applying binary-class cross-entropy loss on the probabilities of target sample x and noise sample $\tilde{x}$, NCE loss is defined:

$$\ell_{NCE} = -\frac{1}{N}\sum_{i}^{N}[log(h_\theta(x_i)) + log(1 - h_\theta(\tilde{x}))] \tag{2.13}$$

With NCE loss, van den Oord et al. proposed **InfoNCE loss** [52] to differentiate the target sample from a list of noise samples, where noise samples will be treated differently by applying multi-class cross-entropy.

Given context c and a sample set $X = \{x_1, x_2, ..., x_N\}$ of $N - 1$ negative samples and one positive sample, the probability of drawing a positive sample from the set can be obtained by:

$$\begin{aligned} p(x^+|X, c) &= \frac{f(x^+, c)}{\sum_{i=1}^{N} f(x_i, c)} \\ where\ f(x, c) &\propto \frac{p(x|c)}{p(x)} \end{aligned} \tag{2.14}$$

$f(x, c)$ is a function that measures density ratio of sample x in context c and expresses the mutual information between x and c. With the probability of a positive sample, infoNCE is obtained by calculating multi-class cross-entropy loss of correctly classifying the positive sample:

$$\ell_{InfoNCE} = -\mathbb{E}[log(p(x^+|X, c))] \tag{2.15}$$

2.3 Discrete Cosine Transform

In computer vision, the typical inputs for CNNs are RGB pixels. An image can be considered as points in the horizontal and vertical directions. The color of each pixel can be decomposed into red, green, and blue elements, i.e., RGB. The color of a pixel is a mixture of the three colors. Typically, two adjacent pixels in an image will be very close in color. Compression is possible if we only store the essential image information rather than every detail of the color alignment. Most modern cameras in HMDs produce high-resolution images, which do not match most CNN input sizes. Instead of aggressively down-sampling the images, the frequency domain of the images provides a better way to preserve more information while reducing the input size. The frequency domain of an image lays the foundation for compression by separating the image into parts differing in information density. The discrete cosine transform (DCT) [1] is the most used method in converting an image to frequency representations.

The DCT (described in Figure 2.6) is similar to Fourier transform as it projects the blocks of color pixels onto the frequency domain. Before DCT, the RGB color space is often converted to YCbCr space, where Y is the luminance component storing brightness information, while Cb and Cr are chroma components representing colors. Each Y, Cb, and Cr channel is divided into 8×8 pixel blocks and undergoes block-wise DCT, where each block produces an 8×8 matrix called DCT coefficients. Each DCT coefficient matrix is a spatial frequency spectrum of an 8×8 image block, and the up-left part is the low-frequency area, and the bottom-right is the high-frequency area. DCT is a no-loss process, and applying inverse DCT will restore the converted image to the color space. In a practical view, down-sampling is often used on Cb and Cr channels since the Y component is often more informative [71]. After DCT, quantization can also be applied by discarding the trivial information in the DCT coefficients.

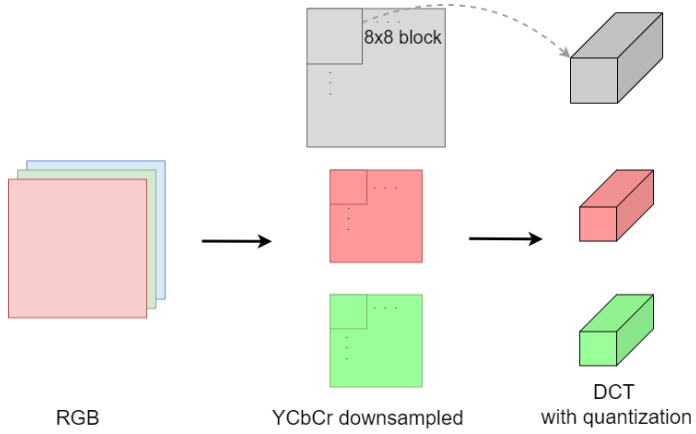


Figure 2.6: **Main process of DCT. The RGB space is converted to YCbCr space. Each component in the YCbCr space will be divided into multiple 8×8 blocks, to which block-wise DCT is applied.**

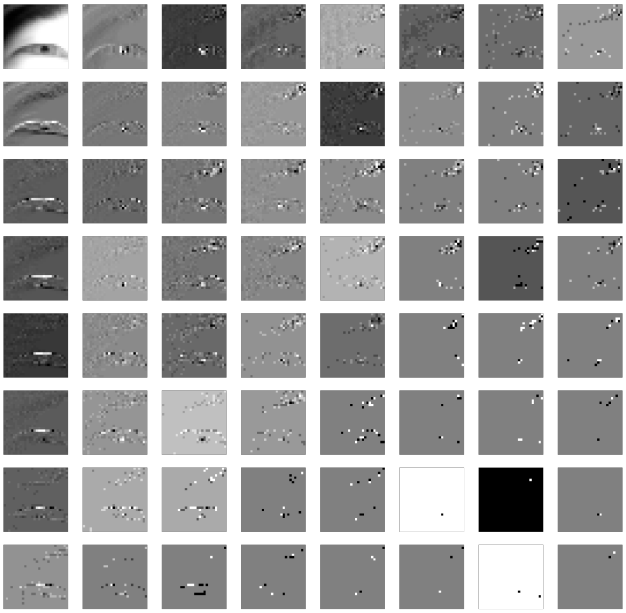


Figure 2.7: **Example visualization of DCT channels for luminance component. The upper left stands for low-frequency channels, while the bottom right stands for high-frequency channels.**

The properties of DCT can be summarized as follows:

- **Energy Compaction**. DCT has proven to be an efficient way to extract essential information due to its energy compaction property [1, 55]: a large portion of signal energy is likely to be concentrated on a subset of the coefficients, especially on the low frequencies. Figure 2.7 visualizes the DCT channels of the luminance component from an example image. We can see that the images expressed by lower frequencies are more discernible than higher frequencies. This property allows it to preserve salient channels that account for major information instead of the whole frequency spectrum. We can remove trivial information with a reduced input size for the training and inference by discarding high-frequency channels.

- **Orthogonality**. DCT can also be regarded as a special process of 8×8 convolution with a stride of 8. The 64 unit filters decompose the pixel blocks into vertical, horizontal, and composite frequency channels. Figure 2.8 shows a visualization of the filters. However, unlike convolutional layers, the filters of DCT are always orthonormal [61] and are not obtained through training, whereas the filters in a convolutional layer are not guaranteed to learn orthonormal filters after training [32]. Based on this property, it is possible to skip some convolutional layers in the models using DCT coefficients as the input. In CNN models with RGB input, the early layers often learn Gabor filters to extract basic features such as edge and texture. Later layers normally learn latent and abstract high-level features [74]. The conversion to the frequency domain through DCT filters behaves similarly to Gabor filters [32]. Thus, it is reasonable to skip some

layers since DCT already serves a similar function as early CNN layers and the output representations are orthonormal. This makes it possible to obtain a smaller model without significant detriment.

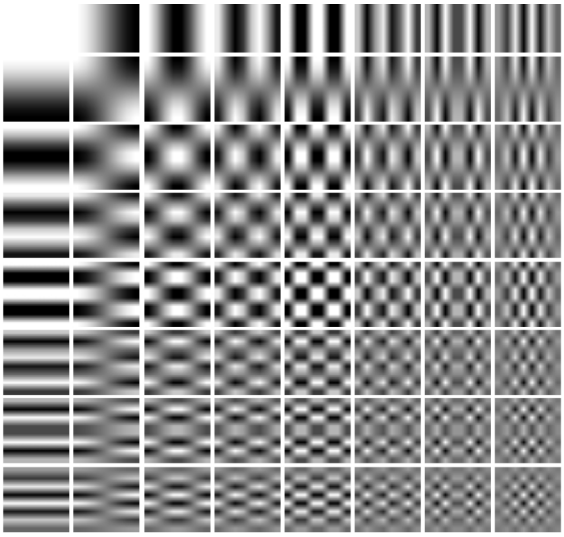


Figure 2.8: **The 64 DCT basic filters for a combination of vertical and horizontal frequencies for an 8×8 image block. Starting from zero, the frequency increases from up to bottom and left to right. The white block (the one in the upper-left corner) represents the DC component.**

Chapter 3

Related Work

This chapter presents related approaches to FER in Section 3.1.1, followed by FER based on periocular images in Section 3.1.2. Section 3.2 reviews related work on training neural networks with image frequency representations.

3.1 Facial Expression Recognition

Facial expression is an indispensable component of interpersonal communication. Existing studies show that linguistic elements comprise one-third of human communication while non-linguistic convey two-thirds [48]. As a popular research field, FER has already been applied in cognitive science, health [4], entertainment [76], and Human-Computer Interaction (HCI) [5], as facial expressions convey visual clues containing implicit messages of human internal state. Various FER systems have been developed to decipher expression-related information from different types of images as input.

3.1.1 Approaches to FER

The conventional FER methods work by means of detecting well-designed expression features. In recent years, more advanced techniques, such as deep learning, have been introduced for automatic feature extraction.

FER with hand-craft features. The conventional FER focused on constructing hand-craft features based on geometric features from face images to train an expression classifier [25, 35]. The common standard to categorize facial expressions based on geometric features is the facial action coding system (FACS) [21]. FACS characterizes facial expressions anatomically as muscle change on the face by encoding facial expressions into a specific combination of movements of action units (AUs), which are the independent definition of expression-related muscles on the human face. The AUs of seven basic facial expressions (Anger, Disgust, Fear, Happiness, Neutral, Sadness, and Surprise) defined by Ekman [20] can be found in Table 3.1. FACS allows us to study facial expressions, collect datasets systematically, and build features, such as local binary patterns (LBP) [59, 77] and histogram of oriented gradient (HOG) [26], for the FER tasks. Even though FER with hand-craft features has been demonstrated to reach good performance, these approaches heavily rely on the

facial expression	AUs
Happy	Cheek Raiser, Lip Corner Puller
Fear	Inner Brow Raiser, Outer Brow Raiser, Brow Lowerer, Upper Lid Raiser, Lid Tightener, Lip Stretcher, Jaw Drop
Sadness	Inner Brow Raiser, Brow Lowerer, Lip Corner Depressor
Surprise	Inner Brow Raiser, Outer Brow Raiser, Upper Lid Raiser, Jaw Drop
Anger	Brow Lowerer, Upper Lid Raiser, Lid Tightener, Lip Tightener
Disgust	Nose Wrinkler, Lip Corner Depressor, Lower Lip Depressor

Table 3.1: **Example of AUs defined by FACS.**

preprocessing step to build discriminative features for a well-performed model, making them often less accurate compared to approaches with automatic feature extraction [22].

FER with automatic feature extraction. Empowered by the advance in deep neural network architectures in recent decades, studies on FER have been put on automatic FER feature extraction. Convolutional neural networks (CNNs) have proven to reach state-of-the-art results in FER tasks [19, 40, 41]. Compared with traditional methods using hand-crafted features, well-designed CNNs can levitate the work on feature engineering by automatically extracting visual representations directly from input images. With effective network structures and sufficient labeled data, deep learning models reach high accuracy and can extract features invariant across datasets for FER tasks [7]. In the field of deep-learning-based FER, several publicly open datasets are used for extensive research and experiments. For instance, the Extended Cohn-Kanade Dataset (CK+) [46] includes video sequences of 123 subjects aged from 18 to 30 years old. The images are 640×480 in resolution. The MUG Facial Expression Database [2] includes RGB 896×896 pixels images taken from recorded videos of 86 subjects. FER2013 [30] contains approximately 30,000 full-facial images with 48×48 resolution. These datasets facilitate the work on building deep learning FER models by including many facial-expression-unrelated variations.

3.1.2 FER with Periocular Images

The common approach for FER is to use full-face images [57]. Due to the increasing popularity of HMDs, exploration has been done to realize FER using periocular images since the models can be trained and implemented using periocular images taken by the cameras already integrated into the devices. Studies show that the periocular area contains emotional information and can be considered a strong indicator of human internal emotional state [45]. The pupil state is shown to be strongly related to the human nervous system and has been associated with cognitive process [6]. The distance between the sclera and iris has been shown to be correlated to emotional state [54]. Based on these indicators, researchers have explored implementing FER using the periocular region. Ghimire et al. [24] divided the full face into different domains and extracted

Figure 3.1: **Exemplar images of seven basic facial expressions from MUG dataset.**

features only from a subset of the regions. They found the eye and mouth regions provide the most discriminant features for facial expressions. Alghowinem et al. showed the possibility of performing expression classification based only on eye movement [3].

However, the periocular areas are sensitively subject to external stimuli. The main challenge of recognizing facial expressions from periocular images is the reduced potential features as we lose input from other facial areas whose movement is also strongly related to the facial expressions. The loss of visual representations makes it more complex to generalize the expression classifiers due to intra-subject variances among users.

Some work has been done to overcome these constraints. Chen et al. [11] reconstruct the whole face to detect facial gestures based on eye and mouth gestures captured by a set of biopotential sensors. Katsutoshi et al. [47] implemented a smart eyewear device that recognizes facial expressions by leveraging sensing skin deformation on the wearer's face. Nie et al. [50] utilize bio-signals from multiple sensors to detect landmarks of facial expression features. However, most of the solutions require additional sensors, demanding more hardware alignment and design work. Some of these sensors are intrusive to human skin, making a long-time use undesirable. In tackling these issues, Hickson et al. [38] demonstrate that eye images captured by the cameras in HMDs already contain enough information to infer a specific set of facial expressions, making it possible to build FER algorithms on HMDs without any additional sensors.

3.2 Learning in the Frequency Domain

Even though FER with RGB images has reached great success, most images taken by HMD cameras have a larger size than the CNN input size. As a substitute for sloppily down-sampling the RGB images, converting representations to frequency domain is effective in filtering out redundant information with little accuracy loss. By applying quantization to the frequency representations, we are able to train a model with a smaller input size and faster training speed.

The quantization process reduces the size of DCT coefficients based on its energy compaction property in the low-frequency channels. The idea has been applied in image compression algorithms such as JPEG. JPEG applies a quantization table to the DCT coefficients, and usually, more weights are put onto low-frequency parts. JPEG is the most widely used image compression standard so far because of its compression ratio unmatched by other traditional compression algorithms without noticeable image quality loss in human eyes. Many datasets are stored in the format of JPEG such as CIFAR-10 [43], ImageNet [16], and MNIST [44].

Conventionally, when our training dataset comprises compressed images, the images will be decoded to arrays of RGB values before being fed into the networks, which takes extra computation in the decoding phase. However, some work has been done in exploring training neural networks by directly using encoded images as the input. Robert et al. bypass the decoding phase by integrating trainable decoder networks into inference networks, and the jointly trained networks reach a comparable performance while heavily reducing computational cost [64]. Wu et al. demonstrate that the compression process eliminates trivial information and can be used for directly training neural networks with accelerated speed [69]. Gueguen et al. explore directly extracting features from DCT coefficients of encoded images. The networks outperform its counterpart with RGB input in speed while remaining the same or even higher accuracy [32]. Xu et al. experimentally prove that in the frequency domain, the luminance component of the YCbCr space and low-frequency channels are higher in information density, and neural networks can learn good representation only from a subset of the DCT coefficients [71].

Chapter 4

Approach

In this chapter, we present the pipeline for training our model, followed by a detailed description of processing in the frequency domain, contrastive learning algorithms, and the design of encoder architectures.

4.1 Overview of Design

The framework of our design is shown in Figure 4.1, which consists of two phases: pre-training and personalized fine-tuning. The encoder is first trained through contrastive learning with unlabeled periocular images. After learning the representations, the trained encoder is transferred as a feature extractor for subject-specific tasks.

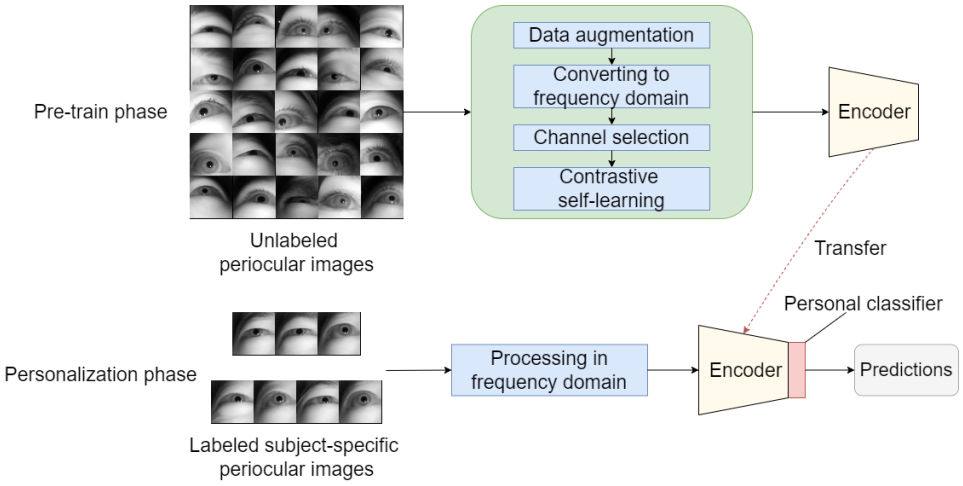


Figure 4.1: **Overview of the training pipeline. In the pre-training phase, a large pool of unlabeled data will be provided as the training set. The pre-trained encoder will be transferred to subject-specific tasks, in which a few labeled samples are used for personalization.**

Pre-training. The unlabeled RGB images are first fed into an image augmentation function to generate positive and negative samples for contrastive

learning. Then, the augmented images are converted into DCT coefficients with 64 frequency channels, based on which a static frequency channel selection is applied by discarding a set of high-frequency channels. After undergoing contrastive learning with the images expressed in the frequency domain, the encoder learns general representations.

Personalized fine-tuning. After the encoder goes through the pre-training stage, it is transferred to the personalization phase as a feature extractor. A dense layer is added to the feature extractor to make predictions of facial expressions. The individual inference models (including the feature extractor and the classification head) are fine-tuned with a small number of labeled subject-specific data to lessen intra-subject variance caused by subject diversity.

4.2 Channel Selection

When the images are converted into the frequency domain through DCT (referring to Section 3.2), we further apply channel selection to prune the input. Figure 4.2 illustrates the channel selection procedure, where high-frequency channels are discarded as low-frequency channels in the DCT coefficients are likely to be more salient.

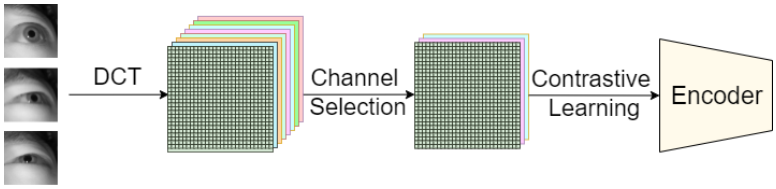


Figure 4.2: **Channel selection**

4.3 Pre-training with Contrastive Learning

This section describes the state-of-the-art contrastive learning methods we use in the thesis: SimCLR, MoCo, and BYOL.

4.3.1 SimCLR

SimCLR [12] is an end-to-end contrastive learning method with a high demand for hardware to support a larger batch size to contrast against more negative samples.

Training

An overview of the training can be found in Figure 4.3.

- **Augmentation**. Data augmentation is first applied for each image in the current mini-batch. We apply geometric and appearance transformation (described in Section 2.2.1) as the augmentation functions to produce positive and negative samples. For the image x, the function produces two

randomly augmented images x_1 and x_2, which are considered a positive pair. Samples derived from other images in the mini-batch are negative.

- **Forward pass**. x_1 and x_2 are then fed into the encoder through forward pass and representations h_1 and h_2 are obtained. The encoder is a CNN model that learns to extract discriminative visual representations during the training. The two visual representations h_1 and h_2 will be forwarded into a two-layer multilayer perceptron (MLP) projection head to further extract features z_1 and z_2.

- **Contrastive loss optimization**. The representations z_1 and z_2 will be computed against negative samples in the current mini-batch for contrastive loss. The encoder is updated with gradients based on the loss.

The projection head is a two-layer MLP (described in Figure 4.4), which includes a batch normalization layer after each dense layer and projects the representations to a lower dimensional space. After training, the projection head will be discarded, and the trained encoder can be used as a feature extractor for downstream tasks.

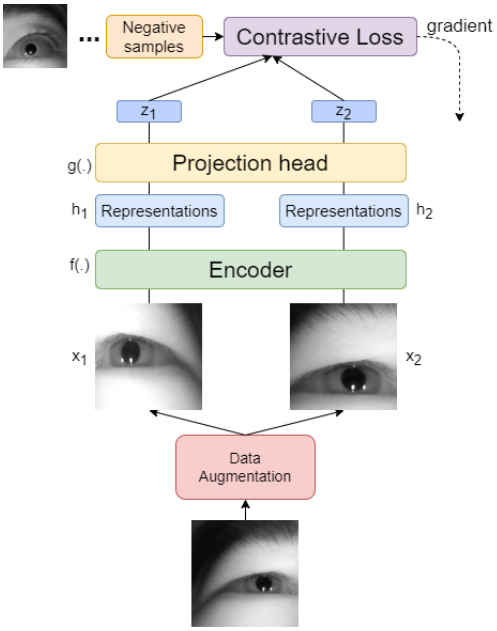


Figure 4.3: **SimCLR**

Loss Function

Given the batch size N, $2N$ data points are obtained after augmentation. For each data point, its augmented counterparts are the positive samples, and the rest $2(N-1)$ samples are regarded as negative. The cosine similarity of z_i and z_j is represented as $cos\_sim(z_i, z_j)$. Based on InfoNCE (Equation 2.15), the loss

function for positive pair(i, j) is defined as:

$$\ell_{i,j} = -log \frac{exp(cos\_sim(z_i, z_j)/\tau)}{\sum_{k=1}^{2N} \mathbb{1}_{[k \neq i]} exp(cos\_sim(z_i, z_k))/\tau}, \tag{4.1}$$

where $\mathbb{1}_{[k \neq i]}$ is a function equal to 1 if $k \neq i$ and is 0 when $k = i$. τ denotes the temperature parameter that controls the distribution concentration of the input vector [70]. The temperature coefficient τ controls how well the model can discriminate against negative samples. If τ is set large, the output logits distribution will show less variance, and the contrast loss tends to treat all negative samples more equally and neglect small differences. In contrast, if the temperature coefficient is set too small, the contrastive loss will focus on the nuanced difference between samples, making the model tend to diverge and perform poorer in terms of generalization.

The total contrastive loss for one mini-batch with size N is defined as:

$$\mathbb{L} = \frac{1}{2N} \sum_{k=1}^{N} [\ell(2k-1, 2k) + \ell(2k, 2k-1)], \tag{4.2}$$

where $\ell(2k-1, 2k)$ denotes the loss of one positive pair. The total loss is termed NT-Xent (Normalized temperature-scaled cross-entropy), computed by averaging the loss of all positive pairs in the mini-batch.

Algorithm 1 Main algorithm for SimCLR

Input: N: batch size; τ: temperature; f: encoder; g: projection head; T: data augmentation;
Output: encoder f(.)

for each minibatch x **do**
  Instantiate two augmentation functions $t_1 \sim T$, $t_2 \sim T$
  $\tilde{x}_1 = t_1(x)$, $\tilde{x}_2 = t_2(x)$
  $h_1 = f(\tilde{x}_1)$, $h_2 = f(\tilde{x}_2)$
  $z_1 = g(h_1)$, $z_2 = g(h_2)$
  for $i \in \{1, 2, ..., 2N\}$ and $j \in \{1, 2, ..., 2N\}$ **do**
    $cos\_sim_{i,j} = (z_i \cdot z_j)/(\|z_i\|\|z_j\|)$
  end for
  Define $\ell_{i,j} = -log \frac{exp(cos\_sim_{i,j}/\tau)}{\sum_{k=1}^{2N} \mathbb{1}_{[k!=i]} exp(cos\_sim_{i,k})/\tau}$
  $\mathbb{L} = \frac{1}{2N} \sum_{k=1}^{N} [\ell(2k-1, 2k) + \ell(2k, 2k-1)]$
  Optimize $\mathbb{L}$ and backpropagate to update f and g
end for

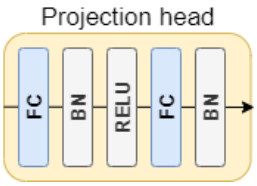


Figure 4.4: **Projection Head in SimCLR**

4.3.2 Momentum Contrastive Learning

Momentum Contrastive (MoCo) learning [36] is a memory-efficient way to learn good visual representations without a large batch size. A larger dictionary can be supported through momentum update on the momentum encoder instead of gradients. Before MoCo, people tended to use a memory bank (described in Section 2.2.2) to keep a large dictionary, which tends to be inconsistent with the newly updated encoder. Moco solves the inconsistency issue by using a queue dictionary and momentum update.

Training

An overview of the training procedure is depicted in 4.5.

- **Augmentation**. For each mini-batch, two augmented views, x_k and x_q, are obtained through the transformation function.
- **Forward pass**. x_k and x_q are respectively forwarded into the key and query encoders that are initialized with the same weights. The extracted key and query are passed into the projection head. Like SimCLR, the projection head is used for further feature extraction and will be discarded after training.
- **Computing loss**. Based on z_k and z_q, which are produced by the projection head, the contrastive loss for the current mini-batch is computed against key representations in the dictionary.
- **Updating encoders**. The query encoder is updated through gradients. The key encoder is updated by weighted average:

$$\theta_k \leftarrow m\theta_k + (1-m)\theta_q, \tag{4.3}$$

  where θ_k denotes parameters of the key encoder, and θ_q demotes those of the query encoder. $m \in [0, 1)$ is the momentum coefficient to determine how fast the key encoder progresses.
- **Updating dictionary**. With the newest z_k, the dictionary is updated by pushing in z_k and removing the oldest representations.

MoCo applies a two-layer projection head (shown in Figure 4.6)to lower the output dimension. It should also be discarded after training, and the trained query encoder can be transferred to other tasks.

MoCo supports a large dictionary, enabling encoders to learn with a large set of negative samples. The loss will show an evident increase at the beginning of training because the dictionary is being filled with new keys, decreasing after enough training epochs. The larger the dictionary size, the more time the dictionary needs to be filled up.

Momentum update makes it possible to generate consistent keys with a small batch size while maintaining a large pool of negative samples. One of the critical parameters in MoCo is the momentum coefficient. To the extreme, it is possible to set the momentum coefficient $m = 1$ [36]: the key and query encoders sharing the weights. The method has proven to fail because of loss oscillation.

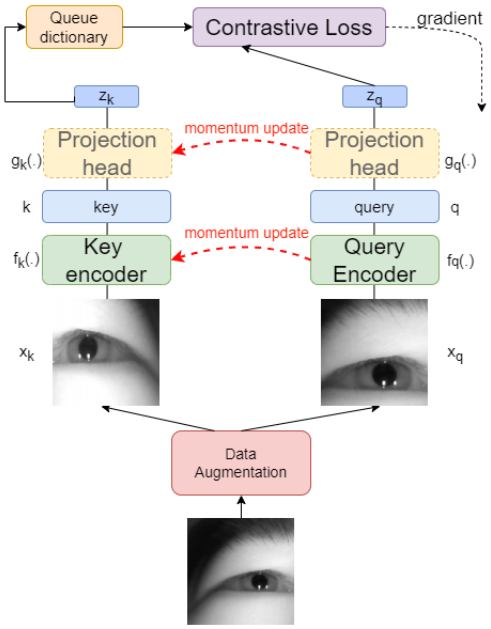


Figure 4.5: **MoCo**

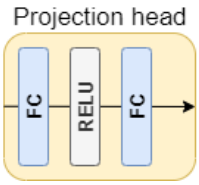


Figure 4.6: **Projection Head in MoCo**

The reason is that the constant and sudden change of parameter values by back-propagation makes the key encoder inconsistent concerning the new mini-batch. It suggests that the smooth progress of the key encoder is beneficial.

Loss Function

MoCo employs InfoNCE (Equation 2.15) as the loss function. For query q (or anchor), the InfoNCE loss is defined:

$$\ell_q = -log \frac{exp(z_q \cdot z_{k+}/\tau)}{\sum_{i=0}^{K} exp(z_q \cdot z_{ki}/\tau)}, \tag{4.4}$$

where z_q represents the normalized representation of query q, z_{k+} denotes the normalized representation of k_+, which is positive to q, and τ is a temperature parameter to control the logits distribution. The dictionary contains $K+1$ representations, including one positive and K negatives. MoCo uses a dot product to measure the similarity of two sample representations.

Algorithm 2 Main algorithm for MoCo

Input: *queue*: queue dictionary τ: temperature; f_k: key encoder; f_q: query encoder; g_k: projection head for key encoder; g_q: projection head for query encoder; m: momentum coefficient; T: data augmentation;
Output: query encoder $f_q(.)$

for each minibatch x **do**
  Instantiate two augmentation functions $t_1 \sim T$, $t_2 \sim T$
  $\tilde{x}_k = t_1(x)$, $\tilde{x}_q = t_2(x)$
  $h_k = f_k(\tilde{x}_k), h_q = f_q(\tilde{x}_q)$
  $k = g_k(h_k), q = g_q(h_q)$
  $k = k.stop\_gradient()$
  $l\_pos = einsum('nc, nc- > n', q, k)$
  $l\_neg = einsum('nc, kc- > nk', q, queue)$
  $logits = concat([l\_pos, l\_neg], axis = 1)$
  $loss = log(softmax(logits/\tau))$
  Optimize loss and backpropagate to update f_q and g_q
  $f_k.params = m * f_k.params + (1 - m) * f_q.params$
  $g_k.params = m * g_k.params + (1 - m) * g_q.params$
  $queue.dequeue(length(k))$
  $queue.enqueue(k)$
end for

4.3.3 Bootstrap Your Own Latent

Most contrastive learning methods train the encoder by optimizing a loss function calculated by comparing positive and negative samples, such as SimCLR and MoCo. However, it is also possible to "bootstrap" the representations from data by "predicting itself" without building negative samples. Bootstrap Your Own Latent (BYOL) [31] is an effective approach for self-supervised learning without building negative samples. It contains two networks, referred to as online and target networks. The online network uses gradients to update weights, while the target network is trained by momentum update. Unlike SimCLR and MoCo, BYOL has an extra prediction layer to predict the output of the online network.

Training

The training process (described in Figure 4.7) is as follows:

- **Augmentation**. Two augmented views x_ξ and x_θ from the anchor are generated.
- **Forward pass**. The augmented data x_ξ and x_θ are fed into target and online networks respectively to get representations: $z_\theta = g_\theta(f_\theta(x_\theta))$ and $z_\xi = g_\xi(f_\xi(x_\xi))$. The prediction $q_\theta(z_\theta)$ is obtained in online network.
- **Computing loss**. The loss is computed based on $\ell 2$ normalized loss of prediction $q_\theta(z_\theta)$ and representation z_ξ
- **Network update**. After the online network is updated with gradients,

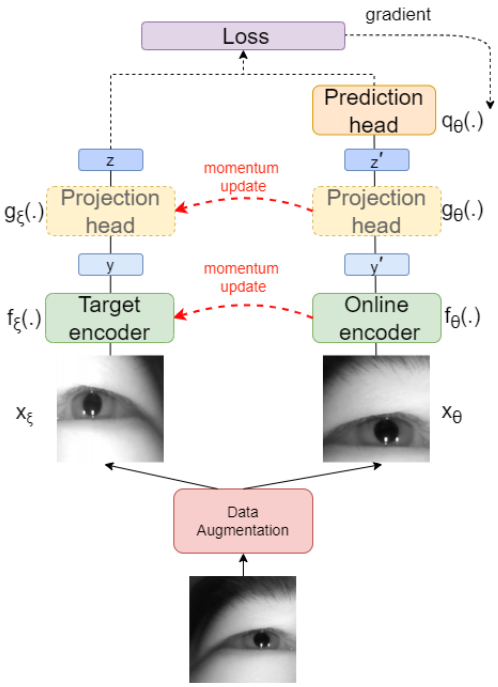


Figure 4.7: **BYOL**

the momentum update is applied to the target network:

$$\xi \leftarrow \tau\xi + (1-\tau)\theta, \tag{4.5}$$

where ξ denotes parameters in the target network and θ those in the online network. The momentum coefficient τ is used to control the update rate.

BYOL uses a two-layer MLP with batch normalization as the project head to project representations to a lower dimensional space. The online network employs a two-layer MLP as the prediction head. Figure 4.8 describes the projection and prediction heads.

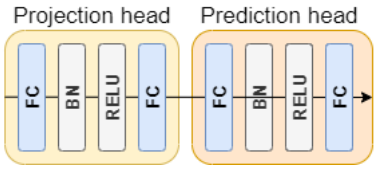


Figure 4.8: **Projection and prediction head in BYOL**

Algorithm 3 Main algorithm for BYOL

Input: f_t: target encoder; f_o: online encoder; g_t: projection head for target encoder; g_o: projection head for online encoder; q_o: prediction head for online encoder; ξ: momentum coefficient; T: data augmentation;

Output: query encoder $f_q(.)$

for each minibatch x **do**
  Instantiate two augmentation functions $t_1 \sim T$, $t_2 \sim T$
  $\tilde{x}_1 = t_1(x)$, $\tilde{x}_2 = t_2(x)$
  $z_t = g_t(f_t(\tilde{x}_1))$, $z_t = z_t.stop\_gradient()$
  $z_o = g_o(f_o(\tilde{x}_2))$, $pred = q_o(z_o)$
  $\overline{z_t} = normalize(z_t)$, $\overline{pred} = normalize(pred)$
  $loss = \|\overline{pred} - \overline{z_t}\|_2^2$
  Optimize loss and backpropagate to update f_o, g_o, and q_o
  $f_t.params = \xi * f_t.params + (1 - \xi) * f_o.params$
  $g_t.params = \xi * g_t.params + (1 - \xi) * g_o.params$
end for

Loss Function

In BYOL, the contrastive loss in calculated using the normalized prediction $\overline{q_\theta}(z_\theta)$ from the online network and the representation $\overline{z_\xi}$ from the target network:

$$\ell_{\theta,\xi} = \|\overline{q_\theta}(z_\theta) - \overline{z_\xi}\|_2^2, \tag{4.6}$$

where $\overline{q_\theta}(z_\theta) = q_\theta(z_\theta)/\|q_\theta(z_\theta)\|_2$, and $\overline{z_\xi} = z_\xi/\|z_\xi\|_2$.

Since no gradient is applied to target parameters ξ, BYOL would avoid converging to a minimum loss of $\ell_{\theta,\xi}$ with regard to both θ and ξ. In other words, the stop-gradient of the target network works as a regularization term to prevent model collapse. Furthermore, a slow momentum update ensures the target is updated while keeping different from the online network, which means the momentum update also serves as a regularization technique.

4.4 Encoder Architectures

We use ResNet-18 as our baseline encoder architecture with RGB input in the thesis. ResNet-18 is a residual network that comprises shortcuts that bypass some network layers [37]. Figure 4.9 visualize an example basic block that incorporates shortcut. The shortcuts enable the gradients to propagate easily to early layers, preventing the neural networks from degradation caused by deepened networks and making it possible to train a deeper network for more difficult tasks.

We modify the original ResNet-18 model with RGB inputs to meet the spatial dimension of DCT inputs. Figure 4.10 illustrates our encoder architectures. For the input data in the frequency domain, only the luminance components are used since the chroma components are zero in value.

The modification principle is to keep the input size comparable to the baseline model while keeping the number of CNNs the same or reduced. The modified encoders form our baseline models in the evaluation experiments.

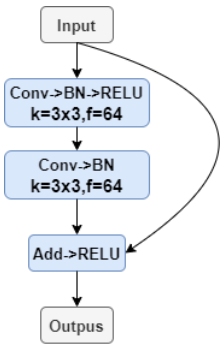
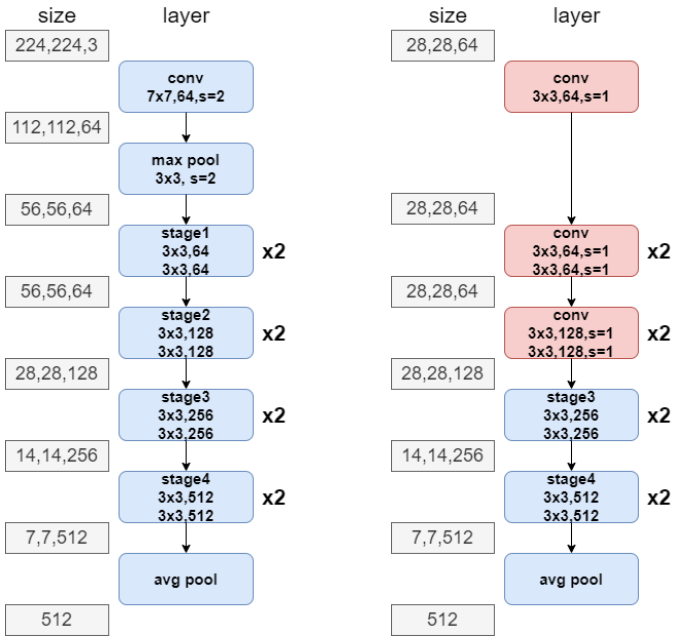


Figure 4.9: **Basic block stage 1 of ResNet-18. A shortcut from the input layer is added to the output layer before the activation function.**



(a) **ResNet-18** (b) **DCT-18**

(c) **DCT-14** (d) **DCT-12**

Figure 4.10: **Encoder architectures. a) ResNet-18 as the baseline encoder with RGB inputs. (The last dense layer is excluded.) b) Encoder for DCT inputs that are comparable to ResNet-18 in terms of the number of convolutional layers. c) Encoder obtained by removing 4 convolutional layers from DCT-18. d) Encoder obtained by removing 2 convolutional layers from DCT-14. For DCT models, the input size displayed in the graph includes all channels. When channel selection is applied, the input size will change accordingly.**

Chapter 5

Evaluation

This chapter presents the evaluation experiments and results using approaches and settings described in Section 4. Section 5.1 describes the experimental settings and implementation details. Section 5.3, 5.4, and 5.5 present the results in terms of the accuracy and the latency performance.

5.1 Methodology

This section describes the approach to collecting and processing data and the tools we use in the evaluation experiments.

5.1.1 Data Preparation

Data Collection

For data collection, 23 volunteers were invited to perform seven basic facial expressions: anger, disgust, fear, happiness, neutral, sadness, and surprise. The volunteers are between 22 and 32, with 13 males and 10 females. They were required to wear a Pupil Core headset shown in Figure 5.1, which has two infrared cameras to record left and right eyes separately. Prior to each recording, an exemplar video was played to the participants as the visual guidance of the corresponding facial expression. Then, they had enough time to experience and practice the facial expression before they were ready to be recorded. The recording lasted for four sessions, where each of the seven basic facial expressions were recorded for 3-4 seconds for 4 times. The recorded videos are 60 FPS with 400×400 pixels. Figure 5.2 shows an example of the data collection experiment.

The data collection aimed to exclude bias and errors as much as possible while keeping the volunteers performing comfortably. To avoid potential bias from instructions provided to the volunteers, we gave little verbal guidance on how to perform the expressions. In addition, to lessen erroneous expressions caused by physical muscle fatigue, the volunteers were allowed to take a break of 10-30 seconds after each recording and 40-60 seconds after each session. They were also allowed to stop at any time during the experiments.

In total, we gathered around 65,000 images labeled with the seven expressions after the data cleaning process. Exemplar images are given in Figure 5.3.

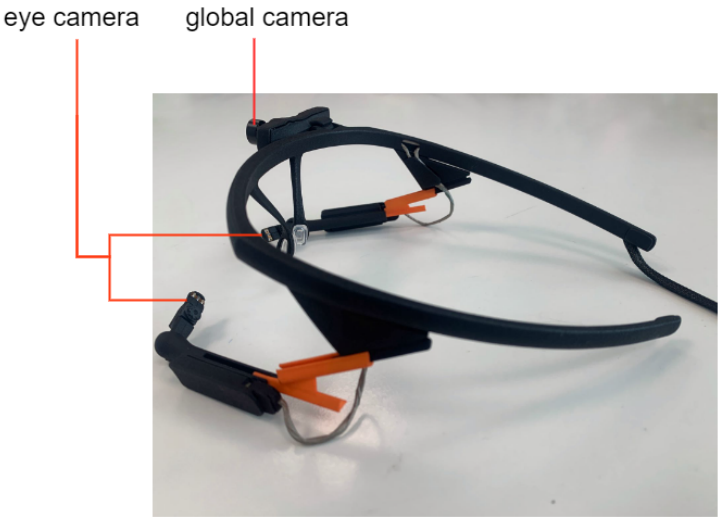


Figure 5.1: **Pupil Lab Core eye-tracker for data collection.**

Figure 5.2: **Data collection experiment**

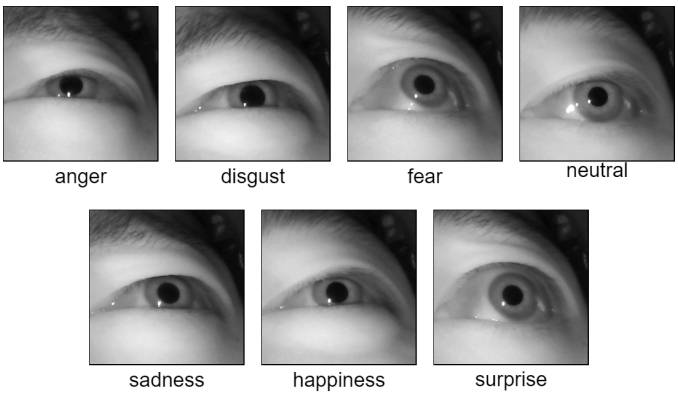


Figure 5.3: **Exemplar eye images for seven facial expressions taken by infrared cameras from the eye-tracker.**

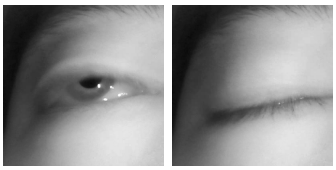


Figure 5.4: **Examplar images of (near)closed eyes to be cleaned**

Data Cleaning

The images in the dataset are obtained by extracting frames from the recorded videos. In order to keep participants relaxed, they were allowed to blink during the recording. To eliminate the variance caused by blinking, we manually remove images of (near)closed eyes from the extracted images. Examples of images to be cleaned are shown in Figure 5.4.

Data Preprocessing

When converting the images to the frequency domain, we only use DCT coefficients of the luminance component since all values in the chroma channels are zero. The images are down-sampled from 400×400 to 224×244 with normalization.

Data from 15 randomly selected subjects are used for pre-training, and the remaining 8 subjects are for fine-tuning the individual inference models. The dataset splitting is illustrated in Figure 5.5. In total, we have approximately 36,000 periocular images in the pre-training stage.

In the personalization phase, the training set is composed of sampled data from two sessions. The remaining two sessions consist of validation and test sets separately. We split the train, test, and validation sets by session to avoid the high similarity between these sets because the images are extracted sequentially from the recorded videos, and images from a video can possibly be distributed to different sets by random splitting. In the fine-tuning stage, a small training set may lead to performance variance with varying selections of samples because small datasets tend to be more biased from the actual data distribution. To smoothen the performance variance, we conduct fine-tuning with five different sampled training sets to calculate the average and quantify the variance using the standard deviation.

As for cross-dataset tasks, we develop a pipeline to crop periocular images from full-face images in the target MUG dataset [2]. An illustration of the pipeline is depicted in Figure 5.6. First, the original RGB images are converted to greyscale, followed by a binarization process where the pixel values above 100 will be set to 255, and the values below 100 will be set to 0. Then, each binarized image undergoes a horizontal projection process, producing a curve representing the added pixel values of the image horizontally as we scan from top to bottom. The curve will show an abrupt decrease as we begin to scan the eye area, and an abrupt increase will be shown as we finish scanning the eye area. Based on this phenomenon, we can select the two peak points in the left part of the curve as the cropping points, with which we crop the image horizontally and obtain an image with two eyes. The cropped images are further fed into a pre-trained eye detection model, and we can obtain two single-eye images from each image.

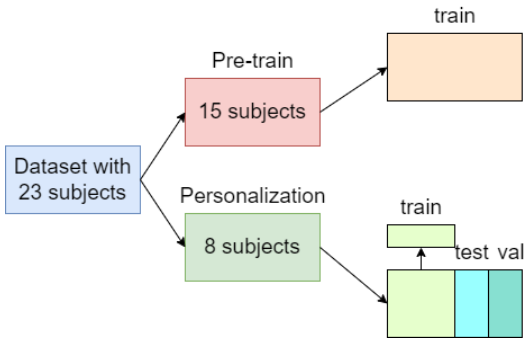


Figure 5.5: **Dataset splitting**

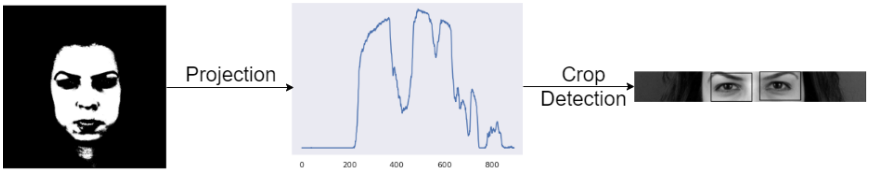


Figure 5.6: **Preprocessing on the MUG dataset: 1. The image is binarized. 2. Horizontal projection is applied to the binarized image, producing a curve representing pixel value horizontally. 3. The image undergoes cropping based on the points selected from the two peaks in the left part of the curve. 4. A pre-trained model detects the eyes from the cropped images.**

This pipeline only works well when the images have a clean background and fixed face position. We observe that the subjects in the MUG dataset are recorded with a relatively fixed position, and the eyes' positions are mainly in the upper half part of the image. In total, we obtain approximately 25,000 periocular images for the cross-dataset task.

5.1.2 Data Augmentation

We apply random crop with resize and color distortion as the data augmentation functions. In the random crop step, the original value for the minimum covered object is 0.1: at least 10% of the original image will be covered after cropping. With this value, the cropped images may only contain trivial features, for example, the upper part of the eyebrows (shown in Figure 5.7). To avoid

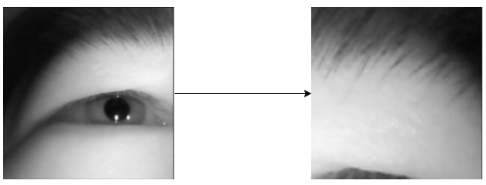


Figure 5.7: **Possible results with the original augmentation function.**

this potential issue, we increase the minimum percentage of the image to be covered to 0.4 to ensure more salient features can be included.

We also observe that some augmentation techniques in the color distortion function do not influence the pixel values of the images. Thus, we remove the unnecessary random saturation, random hue, and color drop from the color distortion step. Examples of the data augmentation are depicted in Figure 5.8.

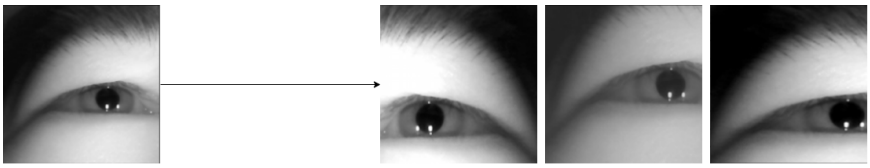


Figure 5.8: **Example results with our augmentation function.**

5.1.3 Implementation and Training

Our models are implemented with Tensorflow and Keras in Python. The data augmentation functions we use in contrastive learning are random crop&resize and color distortions. The processing in the frequency domain is realized by using `jpeg2dct` Library [32], which can directly read DCT coefficients from JPEG images. The specifications of the system where we conduct the experiments are described in Table 5.1 and Table 5.2. Following are the parameters we set for each contrastive learning algorithm:

SimCLR. We use an Adam optimizer with a learning rate of 0.1. The model is trained for 100 epochs with a batch size of 128. The temperature parameter is 0.1.

MoCo. An Adam optimizer with a learning rate of 0.1 is applied. We train the model for 100 epochs with a batch size of 128 and a dictionary size of 6000. The temperature parameter is set to 0.07. The key encoder is updated with a momentum coefficient of 0.999.

BYOL. We apply an Adam optimizer with a learning rate of 0.1. The model is trained for 100 epochs with a batch size of 128. The momentum coefficient is set to 0.999.

In the fine-tuning phase, the models are trained for 500 epochs, and the batch size is the size of the training set. For randomly initialized models without undergoing any pre-training, we fine-tune them for 1000 epochs.

Component	Description
CPU	Intel Xeon(R) Gold 6126 @ 2.60GHz
GPU	NVIDIA GeForce RTX 2080 Ti x 4
Memory	128GB
OS	Ubuntu 18.04.6 LTS

Table 5.1: **System specifications for training**

Component	Description
CPU	Intel(R) i7-8550U @ 1.80GHz
GPU	NVIDIA GeForce 940MX
Memory	8GB
OS	Ubuntu 18.04.6 LTS

Table 5.2: **System specifications for inference**

5.2 Baseline Models

The experiments are conducted by comparing our proposed model-DCT-14-SimCLR-with the baseline models. This section describes the models we use for comparison.

- **ResNet-18-based**. ResNet-18 [37] is a CNN model based on residual blocks. The input type is RGB pixel arrays with the size of $224 \times 224 \times 3$. By comparing our DCT models with ResNet-18, we can evaluate the efficiency of switching to learning in the frequency domain. Based on ResNet-18, we have multiple baseline models that differ in training methods. a). **ResNet-18-supervised** is pre-trained with supervised learning with RGB images. b). **ResNet-18-SimCLR** is pre-trained with SimCLR with RGB images. c). **ResNet-18-random** is randomly initialized without any training.

- **DCT-18-based**. DCT-18 is a DCT model with the same number of convolutional layers as ResNet-18. It is implemented to compare varying input domains (RGB or DCT) and to evaluate how much performance loss there will be if we remove some layers from DCT-18. The baseline models based on DCT-18 include a). **DCT-18-supervised** is pre-trained with labeled data. b). **DCT-18-SimCLR** uses SimCLR as the pre-training method .

- **DCT-14-based**. DCT-14 is developed by removing 4 convolutional layers from DCT-18. We implemented several baselines based on DCT-14: a). **DCT-14-supervised** is trained with supervision. b). **DCT-14-random** is randomly initialized without training. c). **DCT-14-MoCo** is trained with MoCo. d). **DCT-14-BYOL** is trained with BYOL.

- **DCT-12-based**. We remove 2 convolutional layers from DCT-14 to obtain DCT-12. The baseline based on DCT-12 is **DCT-12-SimCLR**, which is pre-trained through SimCLR.

When learning in the frequency domain, we apply different channel selection strategies to prune the input DCT coefficients (visualized in Figure 5.9). The strategies are described as follows:

- **Direct Current (DC) Channel**. DC channel is channel 0, whose frequency is zero. We assume the DC channel contains a large portion of information about the original image. This can be tested by comparing the DC channel with other channels.

0	1	5	6	14	15	27	28
2	4	7	13	16	26	29	42
3	8	12	17	25	30	41	43
9	11	18	24	31	40	44	53
10	19	23	32	39	45	52	54
20	22	33	38	46	51	55	60
21	34	37	47	50	56	59	61
35	36	48	49	57	58	62	63

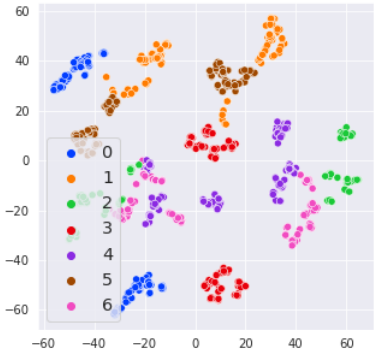
Figure 5.9: **Channel selection strategies. a). DC channel: channel 0. b). Channel 0,1,2: the upper-left triangle that includes the three lowest frequencies. c). Channel 0-5: the upper-left triangle that includes the six lowest frequencies. d). AC channels: all channels excluding the DC channel.**

- **Channel 0,1,2**. Channels 0, 1, and 2 are placed in the upper-left triangle of the DCT coefficient matrix. The triangle represents the three channels with the lowest frequency and may include most of the critical information.
- **Channel 0-5**. We extend the upper-left triangle to include the five lowest frequency channels for better comparison
- **Alternating Current (AC) Channel**. AC channels include all channels except DC channels. We can test whether the DC channel is critical by comparing AC channels with all channels.

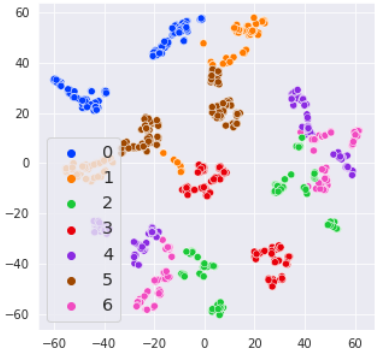
5.3 Representation Visualization

To better understand whether the models have learned good features through contrastive learning, we employ t-distributed Stochastic Neighbor Embedding (t-SNE) [66] to visualize the high-dimensional feature maps for the pre-trained models without undergoing the personalization phase. To achieve this, we randomly sample 2000 images from one subject not included in the pre-training phase and feed their frequency representations into different models. The output dimension of the models is 512. Figure 5.10a and Figure 5.10b depict the original samples.

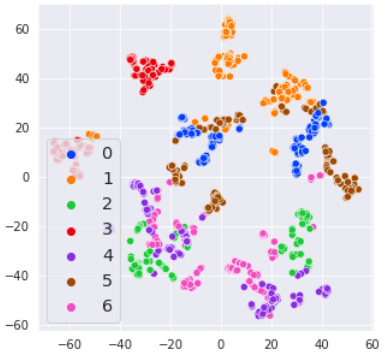
The models we select for t-SNE visualization are DCT-14-supervised and DCT-14-SimCLR, and channel selection is applied to these models. Both the projection and prediction heads are removed before extracting prior to the visualization. Figure 5.10 displays the feature map visualization of the models. It is observed that in supervised models, the clusters tend to overlap with each other in the representation space. In comparison, the clusters in the SimCLR models tend to be separated. Interestingly, we notice that the representations



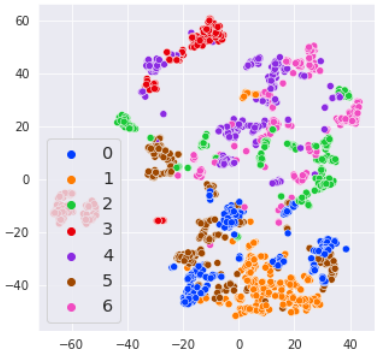
(a) **Original, channel 0,1,2**



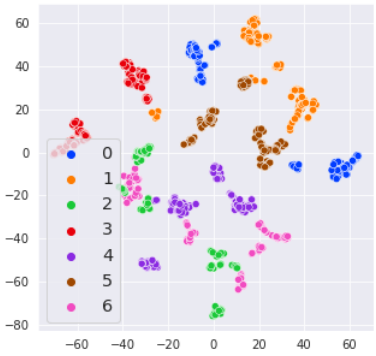
(b) **Original, channel 0-5**



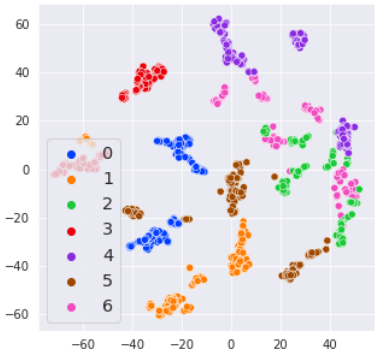
(c) **DCT-14-sup, channel 0,1,2**



(d) **DCT-14-Sup, channel 0-5**



(e) **DCT-14-SimCLR, channel 0,1,2**



(f) **DCT-14-SimCLR, channel 0-5**

Figure 5.10: **t-SNE Visualization of the original data sample and the learned feature maps by supervised learning and SimCLR. Legend: 0-anger, 1-disgust, 2-fear, 3-happiness, 4-neutral, 5-sadness, 6-surprise.**

of 2 (fear) and 6 (surprise) tend to be close in the visualization. This aligns with our understanding that these expressions are similar in appearance and have shared periocular AUs based on the facial action coding system described in Table 3.1.

5.4 Evaluation of Accuracy

This section presents the evaluation results of prediction accuracy in different scenarios, including varying the personalization sample size, channel selection strategies, input domain (RGB versus frequency representations), encoder architectures, and contrastive learning algorithms.

5.4.1 Performance overview

Table 5.3 gives an overview of the performance comparison of DCT-14-SimCLR with other baselines. The sample size in the fine-tuning state is 20. SimCLR models show relatively equivalent results with the supervised model while the number of labels used is reduced from approximately 36,000 to 140 (about 99% reduction). When randomly initialized, DCT-14 has the worst performance and is improved by approximately 50% after pre-trained with SimCLR. It is also shown that DCT-14-SimCLR has reached a comparable performance with bigger models (ResNet-18 and DCT-18), suggesting a smaller model is sufficient for the FER task. We also observe that the performance gap between DCT-18-SimCLR and ResNet-18-SimCLR is very marginal, and there is an insignificant performance drop when only using channels 0, 1, 2 instead of all channels in DCT-14-SimCLR. The result suggests that utilizing frequency representations and applying the channel selection strategies do not generate significant performance loss.

	Input	Accuracy (%)
DCT-14-SimCLR	DCT 0,1,2	83.1
DCT-14-SimCLR	DCT all	83.5
DCT-14-Random	DCT all	30.8
DCT-12-SimCLR	DCT 0,1,2	80.7
DCT-18-SimCLR	DCT 0,1,2	83.6
ResNet-18-supervised	RGB	86.3
ResNet-18-SimCLR	RGB	83.8
ResNet-18-random	RGB	78.3

Table 5.3: **Performance comparison with personalization sample size of 20.**

5.4.2 Varying Personalization Sample Size

We investigate how the models perform with different sample sizes in the fine-tuning stage. The number of samples for each expression class ranges from 2 to 50. The models are fine-tuned with five training sets for each sample size and obtain the average accuracy. The results, depicted in Figure 5.11, are

the average accuracy results of 8 subjects and the standard deviation in the different fine-tuning training sets. We observe that the models will generally perform better as we increase the sample size because the training set brings more features and less bias for the learning. The performance may drop as more data are added to the training set. This may be due to a covariance shift caused by newly added data. Supervised learning with RGB input reaches the highest performance, while the DCT model without pre-training has the worst performance. In Figure 5.11b, we can see that our data augmentation function performs better than the one with original parameters. Figure 5.11c depicts the standard deviation with respect to different sample sizes. The performance deviation decreases as we increase the training sample size. Another observation is that the models with RGB inputs show greater decreases in variance than their DCT counterparts. We choose the sample size of 20 for the rest of the evaluation experiments since the performance increase becomes marginal at this poin

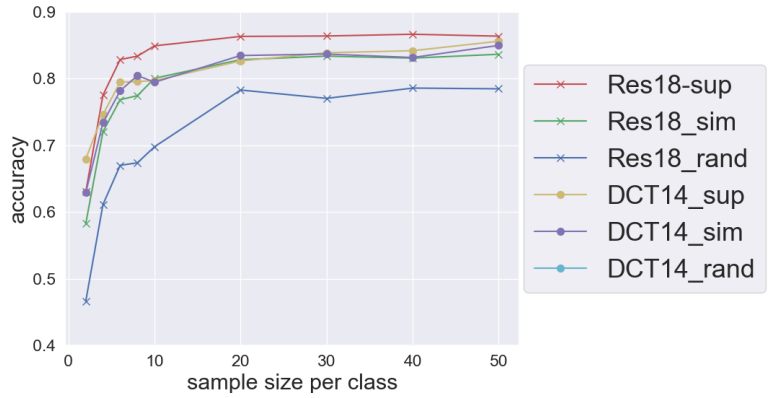
5.4.3 Varying Input Channels

The ResNet-18 encoder has an input size of 224×224 with three RGB channels. The values in the three RGB channels are the same for each image in our dataset. We want to test if it is possible to modify the input size into $224 \times 224 \times 1$ and only use the unique values in one channel. We run experiments to see the performance of the two models. It can be observed in the results shown in Table 5.4 that the RGB model with three channels performs better than that with only one channel.

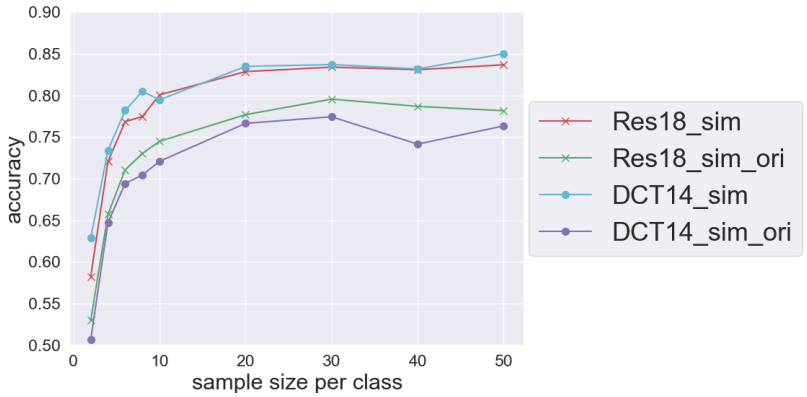
Number of channel(s)	3	1
Accuracy(%)	83.8	81.9
std	0.023	0.029

Table 5.4: **Results varying RGB channel number. Encoder: ResNet-18-SimCLR.**

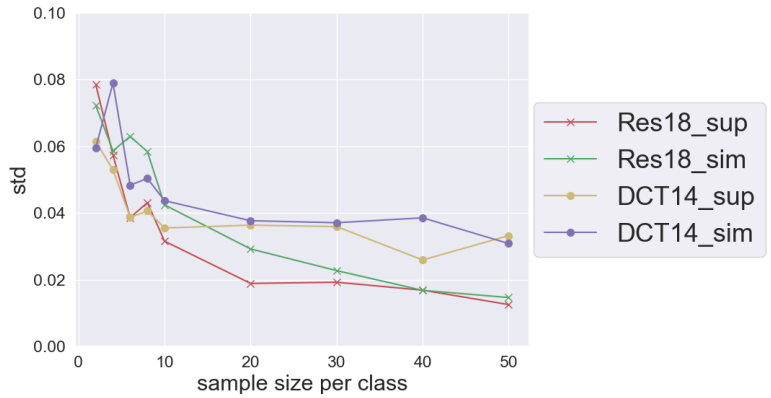
In the frequency domain, low-frequency channels tend to contain more crucial information about the original image. Channel selection is applied by preserving critical channels to reduce the data size. We run experiments to probe the performance sacrifice with different channel selection strategies. An overview of the results is shown in Figure 5.12 and Table 5.5. Comparing AC with all channels, we observe that when the DC channel is removed, there is a significant performance drop, suggesting the DC channel has the most information for the classification task. As more low-frequency channels are included, the accuracy tends to increase, but the increase is gradually marginal and becomes insignificant when sufficient channels are included. In terms of standard deviation, it is observed that the DC channel alone brings higher performance variation, and the variation becomes smaller as more channels are incorporated. The performance variation remains high when the DC channel is not included.



(a) **Accuracy comparison of different pre-training methods**



(b) **Accuracy comparison of different augmentation**



(c) **Standard deviation comparison**

Figure 5.11: **Performance comparison in the personalization stage w.r.t varying sample size. The RGB models are ResNet-18 while the DCT models are based on DCT-14. Notation: sup-supervised, sim-SimCLR, rand-random, ori-original data augmentation. The DCT models are trained with all frequency channels.**

Channel selection	DC	0,1,2	0-5	AC	All
Accuracy(%)	78.2	83.1	83.6	78.0	83.5
std	0.064	0.041	0.039	0.0531	0.038

Table 5.5: **Results of channel selection in the frequency domain. Encoder used: DCT-14-SimCLR.**

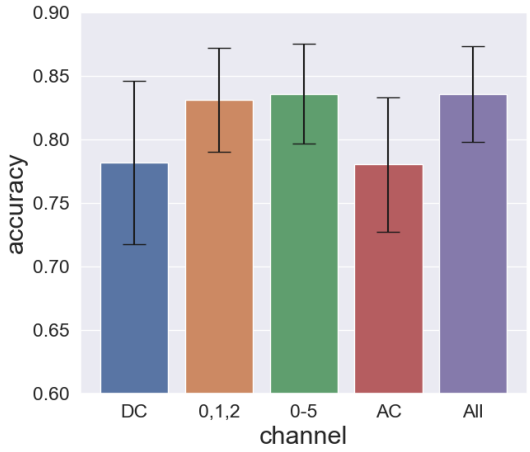


Figure 5.12: **Result comparison of different channel selection strategies in the frequency domain. The model used is DCT-14-SimCLR. The training sample size is 20 images per class.**

5.4.4 Varying Input Domain

We investigate the performance difference when switching from training with RGB pixels to frequency channels. Table 5.6 and Figure 5.13 compare the performances of the models trained in the RGB and frequency domains. It can be observed that the model trained with RGB pixels has better performance and less variance. Models trained in DCT channels also perform relatively well when pre-trained with supervision or contrastive learning.

	Input	Accuracy (%)	std
ResNet-18-supervised	RGB	86.3	0.019
DCT-18-supervised	DCT 0,1,2	82.1	0.031
DCT-18-supervised	DCT 0-5	83.2	0.024
ResNet-18-SimCLR	RGB	83.8	0.023
DCT-18-SimCLR	DCT 0,1,2	83.6	0.035
DCT-18-SimCLR	DCT 0-5	83.9	0.029

Table 5.6: **Results with different input domains.**

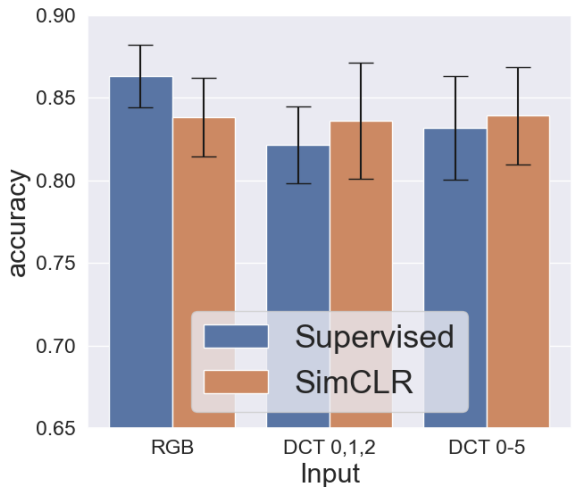


Figure 5.13: **Result comparison of different input domains.**

5.4.5 Varying Encoder Architectures

We compare encoders with different architectures varying in the number of convolutional layers. The encoders are designed by removing layers from ResNet-18 and are pre-trained with SimCLR. The results are displayed in Table 5.7 and Figure 5.14. It suggests a bigger encoder brings higher accuracy and less variance in performance. However, the benefits are marginal after a threshold, implying a smaller DCT model is sufficient to perform the task. Besides the performance gain by the bigger models, the increased training and inference time should also be considered a trade-off. We will explore the response latency in Section 5.5.

	Input	Accuracy(%)	std
DCT-12-SimCLR	DCT 0,1,2	80.7	0.052
DCT-14-SimCLR	DCT 0,1,2	83.1	0.041
DCT-18-SimCLR	DCT 0,1,2	83.6	0.035
DCT-12-SimCLR	DCT 0-5	81.0	0.049
DCT-14-SimCLR	DCT 0-5	83.5	0.038
DCT-18-SimCLR	DCT 0-5	83.9	0.024

Table 5.7: **Results of different encoder architectures.**

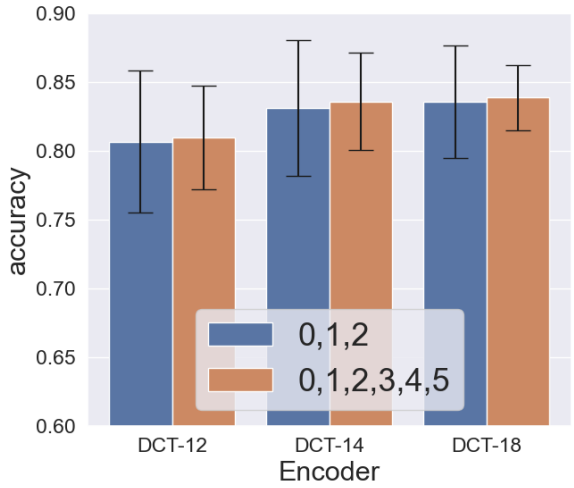


Figure 5.14: **Result comparison of different encoder architectures.**

5.4.6 Varying Contrastive Learning Algorithms

There are different algorithms for contrastive learning. SimCLR is an end-to-end method that learns by comparing samples in the current batch. MoCo uses momentum update and a queue dictionary to support a larger pool of negative representations. BYOL learns without negative samples by predicting representations in a bootstrap manner. We run experiments with different choices of contrastive techniques to see whether similar performance can be realized with a different algorithm. Figure 5.15 compares the accuracy and standard deviation of the models pre-trained with the three contrastive learning methods. We observe that SimCLR has the highest accuracy, and BYOL also performs well. MoCo has the lowest performance variance. The results show that our approach is model-agnostic and can work with different contrastive learning algorithms.

	Input	Accuracy (%)	std
DCT-14-SimCLR	DCT 0,1,2	83.1	0.041
DCT-14-MoCo	DCT 0,1,2	70.1	0.031
DCT-14-BYOL	DCT 0,1,2	80.0	0.043
DCT-14-SimCLR	DCT 0-5	83.5	0.038
DCT-14-MoCo	DCT 0-5	71.8	0.023
DCT-14-BYOL	DCT 0-5	80.7	0.040

Table 5.8: **Results of differen contrastive learning methods.**

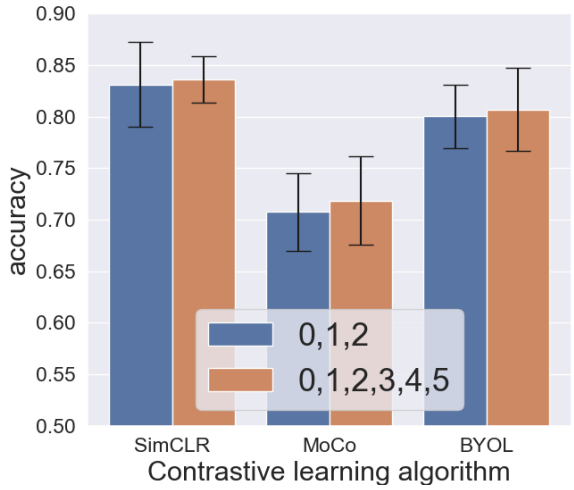


Figure 5.15: **Result comparison of different contrastive learning algorithms. Encoder used: DCT-14.**

	Input	
Model	DCT 0,1,2	DCT 0-5
DCT-14-random	61.78	73.86
DCT-14-SimCLR	94.90	95.64
DCT-14-Supervised	83.78	87.08

Table 5.9: **Accuracy (%) on cross-dataset tasks. Source dataset: our collected dataset. Destination dataset: MUG dataset.**

5.4.7 Cross Dataset

To investigate how our models perform in terms of generalization, we run experiments on cross-dataset tasks. A DCT-14 encoder is pre-trained on our collected dataset and fine-tuned and tested on the MUG dataset. Table 5.9 shows the results on the personalized fine-tuning tasks. It is observed that SimCLR has higher accuracy on the task than supervised learning, suggesting that contrastive learning can learn features that are transferable on different datasets. The performance discrepancy indicates that contrastive learning shows better generalizability than supervised learning in the task.

5.5 Evaluation of Latency

The merit of learning in the frequency domain is reducing computational load with trivial accuracy loss. On the one hand, this technique shows practical potential in accelerated training and faster inference response. On the other hand, this allows us to conduct fine-tuning or inference on a platform where RGB inputs cannot work because of limited hardware resources. To evaluate the speed gain provided by channel selection, we investigate the fine-tuning

time and inference time on a local machine. As a basic measurement of model complexity, we list each architecture's floating point operations (FLOPs) in Table 5.10.

Encoder	ResNet-18		DCT-18	
Input size	224,224,3	224,224,1	28,28,64	28,28,3
FLOPs (G)	1.82	1.74	1.40	1.37

Encoder	DCT-14		DCT-12	
Input size	28,28,64	28,28,3	28,28,64	28,28,3
FLOPs (G)	1.10	1.07	0.86	0.84

Table 5.10: **FLOPs of the encoders**

5.5.1 Fine-tuning Time

The training time is tested with a fixed dataset with two samples per class, i.e., 14 images. All models are trained for 100 epochs. Table 5.11 and Figure 5.16a compare our DCT model with the baseline RGB model. Compared with ResNet-18, we see DCT-18 is approximately 1.58 faster in training time. This implies that training in the frequency domain takes much less time than in RGB space. The training time gradually decreases when the models become smaller. DCT-14 can be 1.71 times faster, while DCT-12 is 2.19 times faster than the baseline model.

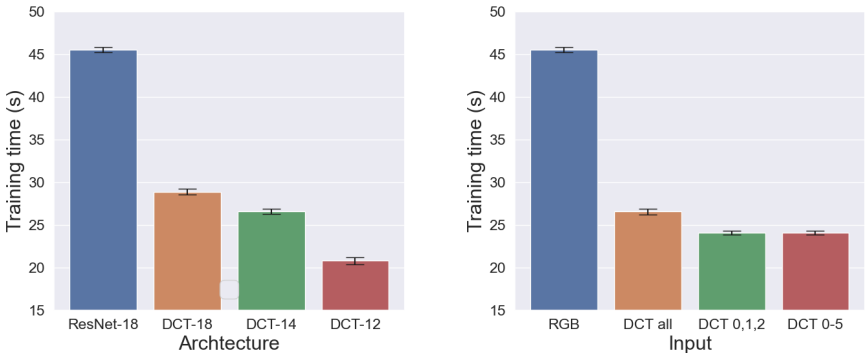
	ResNet-18	DCT-18 all	DCT-14 all	DCT-12 all
Load	0.03	0.02	0.02	0.02
Train	45.49	28.83	26.54	20.78
Sum	45.52	28.86	26.56	20.80

Table 5.11: **Training time (s) comparison for different encoders.**

	RGB	DCT-14 all	DCT-14 0,1,2	DCT-14 0-5
Load	0.03	0.02	0.02	0.02
Train	45.49	26.54	24.06	24.06
Sum	45.52	26.56	24.08	24.08

Table 5.12: **Training time (s) comparison for different inputs.**

Table 5.12 and Figure 5.16b show the result of fine-tuning time for DCT-14 model with different channel selection strategies in the frequency domain. With channels 0,1,2 or channels 0-5 as input, the fine-tuning is approximately 1.89 times faster than the RGB baseline model.



(a) **Training time varying encoders.** (b) **Training time varying inputs.**

Figure 5.16: **Comparison of training time (data loading time included) for 100 epochs with a batch size of 14. Encoder used for DCT models in (b): DCT-14.**

5.5.2 Inference Time

The inference time is recorded while the models run inference for a single image 10,000 times. Data loading time and the actual inference time are recorded separately. The given results are the inference time for a single image on average.

Table 5.13 and Figure 5.17a display the comparison result of inference time with different CNN encoders. We observe that DCT-18 is 1.14 faster than the ResNet-18 baseline model. The inference time can be further reduced by making the architecture smaller. DCT-14 is 1.21 faster than the baseline, while DCT-12 is 1.26 faster.

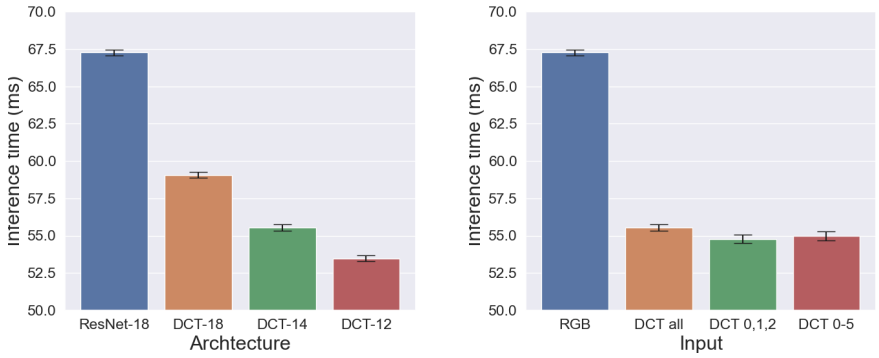
	ResNet-18	DCT-18 all	DCT-14 all	DCT-12 all
Load	3.12	1.17	1.09	1.10
Inference	64.14	57.90	54.45	52.37
Sum	67.25	59.07	55.54	53.47

Table 5.13: **Inference time (ms) of different encoders.**

If channel selection is applied in the frequency domain, the inference time can be further reduced, referring to Table 5.14 and Figure 5.17b. If we switch from ResNet-18 with RGB to DCT-14 with frequency channels 0,1,2, the inference speed can be 1.23 faster.

	RGB	DCT-14 all	DCT-14 0,1,2	DCT-14 0-5
Load	3.12	1.09	0.93	0.93
Inference	64.14	54.45	53.85	54.03
Sum	67.25	55.54	54.78	54.96

Table 5.14: **Inference time (ms) with different inputs.**



(a) **Inference time varying encoders.** (b) **Inference time varying inputs.**

Figure 5.17: **Comparison of inference time with a single image as input, data loading time included. Encoder used for DCT models in (b): DCT-14.**

5.6 Overall Comparison

With the results presented in the previous sections, we can compare our proposed model with the baseline models. A trade-off will be considered in terms of accuracy and latency. Figure 5.18 depicts the results in a 2-D plane. The time in the x-axis is the average result of the training time and inference time after min-max normalization. It is observed that ResNet-18 with supervised learning in the RGB domain has a higher accuracy while the latency performance is the worst. DCT-12 models have the best latency results while performing poorly in accuracy. A good trade-off will be sacrificing some accuracy performance for a big acceleration boost. DCT-14 encoder with frequency channels of 0,1,2 and 0-5 give a good result when pre-trained with SimCLR compared to other baseline models.

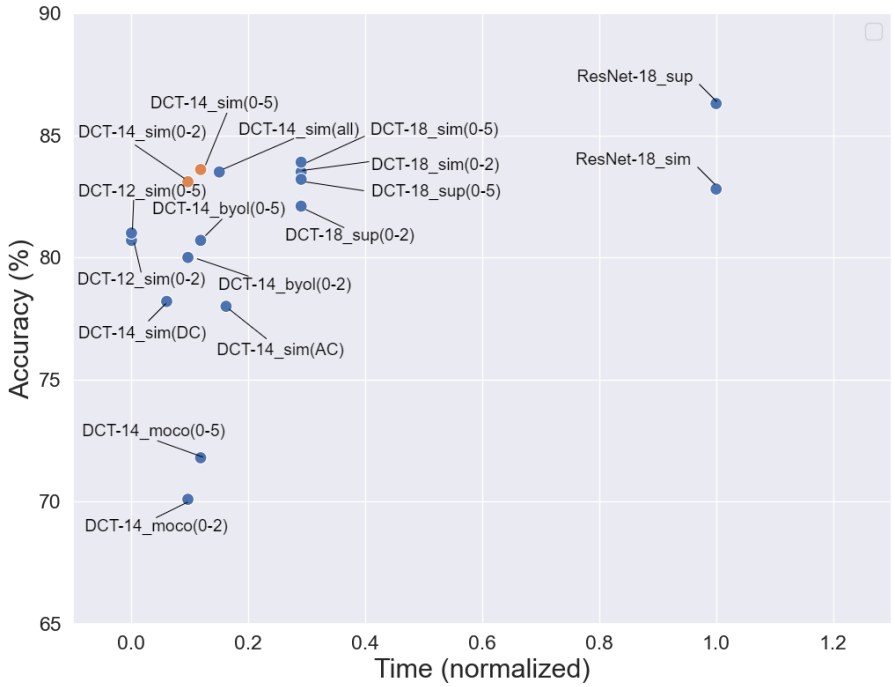


Figure 5.18: **Overall comparison as to accuracy and latency.**

Chapter 6

Conclusion and Discussion

6.1 Conclusion

In the thesis, we apply contrastive learning techniques to learn representations from unlabeled periocular images for FER tasks. In addition, we train the encoders using a subset of the image frequency representations instead of the conventional RGB pixel arrays to reduce the input size and accelerate the encoder in the fine-tuning and inference stages. For the evaluation, We collect a dataset of periocular images labeled with emotional expressions, and we conduct several comparison experiments to validate the effectiveness of our proposed methods.

Our contrastive learning models have reached a reasonably good performance only using a small scale of labeled data, saving massive manual annotation work required by supervised learning. Models trained with frequency input have comparable performance with their RGB counterparts, whereas the training and inference time show a promising reduction. By comparing different encoder architectures in the frequency domain, we also show that a smaller encoder is sufficient to perform the FER tasks. This implies that removing some CNN layers from the frequency models is reasonable.

6.2 Future Work

Contrastive learning models have given promising results in facial expression recognition using the periocular image. However, there is still room for improvement in terms of usability and generalizability. We have proposed the following topics for future study:

- **Diversity of dataset for pre-training**. We only collected data from 23 volunteers to validate our proposed model. A more diverse dataset enables contrastive learning to exclude expression-irrelevant features such as age, skin color, ethnicity, etc. Thus, more data are needed for a larger pool of subjects if we consider designing an engineering or commercial product.
- **Dynamic channel selection**. Although, in general, low-frequency channels are likely to be more informative, some high-frequency channels may contain information that is beneficial for the task [71]. Unlike static channel selection strategies, a dynamic channel selection strategy can help

quantitatively identify each channel's importance in the specific task. Dynamic channel selection adds a tensor that outputs a binary value to control each channel's "on-off" state. The times of being switched to "on" represent the channel's importance. This technique can estimate channel importance based on which static channel selection can be applied, allowing us to adjust the channel selection strategy to a specific task.

- **Versatility**. Besides facial-expression-responsive avatars in a remote VR meeting, facial expression detection can be used for other purposes, such as authentication assisted with facial expressions. If our model is incorporated into an embedded platform, signals from other built-in sensors can be combined to create multiple possibilities. For example, inertial measurement unit (IMU)-based gesture detection and camera-based FER can jointly generate a smiling and nodding avatar for VR meeting on HMDs.

- **Detection of irrelevant events**. In a realistic view, blinking is unavoidable for users of HMDs. Predicting facial expressions using a blinking eye image will cause a false result. Since our model is sensitive to samples in the personalization stage, images of blinking eyes are likely to make the model mix up the prediction classes. Additional steps to avoid unwanted events will increase the usability of the FER platform as an engineering product.

Bibliography

[1] Nasir Ahmed, T_ Natarajan, and Kamisetty R Rao. Discrete cosine transform. *IEEE transactions on Computers*, 100(1):90–93, 1974.

[2] Niki Aifanti, Christos Papachristou, and Anastasios Delopoulos. The mug facial expression database. In *11th International Workshop on Image Analysis for Multimedia Interactive Services WIAMIS 10*, pages 1–4, 2010.

[3] Sharifa Alghowinem, Majdah AlShehri, Roland Goecke, and Michael Wagner. Exploring eye activity as an indication of emotional states using an eye-tracking sensor. In *Intelligent systems for science and information*, pages 261–276. Springer, 2014.

[4] Mohammad Amin Assari and Mohammad Rahmati. Driver drowsiness detection using face expression recognition. In *2011 IEEE international conference on signal and image processing applications (ICSIPA)*, pages 337–341. IEEE, 2011.

[5] Christoph Bartneck and Michael J Lyons. Hci and the face: Towards an art of the soluble. In *International Conference on Human-computer Interaction*, pages 20–29. Springer, 2007.

[6] Margaret Bradley, Laura Miccoli, Miguel Escrig, and Peter Lang. The pupil as a measure of emotional arousal and autonomic activation. *Psychophysiology*, 45:602–7, 08 2008.

[7] Ran Breuer and Ron Kimmel. A deep learning perspective on the origin of facial expressions. *arXiv preprint arXiv:1705.01842*, 2017.

[8] Peter Burkert, Felix Trier, Muhammad Zeshan Afzal, Andreas Dengel, and Marcus Liwicki. Dexpression: Deep convolutional neural network for expression recognition. *arXiv preprint arXiv:1509.05371*, 2015.

[9] Mathilde Caron, Piotr Bojanowski, Armand Joulin, and Matthijs Douze. Deep clustering for unsupervised learning of visual features. In *Proceedings of the European conference on computer vision (ECCV)*, pages 132–149, 2018.

[10] Mathilde Caron, Ishan Misra, Julien Mairal, Priya Goyal, Piotr Bojanowski, and Armand Joulin. Unsupervised learning of visual features by contrasting cluster assignments. *Advances in Neural Information Processing Systems*, 33:9912–9924, 2020.

[11] Chen Chen, Ke Sun, and Xinyu Zhang. Exgsense: Toward facial gesture sensing with a sparse near-eye sensor array. In *Proceedings of the 20th International Conference on Information Processing in Sensor Networks (co-located with CPS-IoT Week 2021)*, pages 222–237, 2021.

[12] Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Everest Hinton. A simple framework for contrastive learning of visual representations. 2020.

[13] Ting Chen, Simon Kornblith, Kevin Swersky, Mohammad Norouzi, and Geoffrey E Hinton. Big self-supervised models are strong semi-supervised learners. *Advances in neural information processing systems*, 33:22243–22255, 2020.

[14] Xinlei Chen and Kaiming He. Exploring simple siamese representation learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 15750–15758, 2021.

[15] Sumit Chopra, Raia Hadsell, and Yann LeCun. Learning a similarity metric discriminatively, with application to face verification. In *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, volume 1, pages 539–546. IEEE, 2005.

[16] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

[17] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[18] Carl Doersch, Abhinav Gupta, and Alexei A Efros. Unsupervised visual representation learning by context prediction. In *Proceedings of the IEEE international conference on computer vision*, pages 1422–1430, 2015.

[19] Samira Ebrahimi Kahou, Vincent Michalski, Kishore Konda, Roland Memisevic, and Christopher Pal. Recurrent neural networks for emotion recognition in video. In *Proceedings of the 2015 ACM on international conference on multimodal interaction*, pages 467–474, 2015.

[20] Paul Ekman. The argument and evidence about universals in facial expressions. *Handbook of social psychophysiology*, 143:164, 1989.

[21] Paul Ekman and Wallace V Friesen. Facial action coding system. *Environmental Psychology & Nonverbal Behavior*, 1978.

[22] Mariana-Iuliana Georgescu, Radu Tudor Ionescu, and Marius Popescu. Local learning with deep and handcrafted features for facial expression recognition. *IEEE Access*, 7:64827–64836, 2019.

[23] Mathieu Germain, Karol Gregor, Iain Murray, and Hugo Larochelle. Made: Masked autoencoder for distribution estimation. In *International conference on machine learning*, pages 881–889. PMLR, 2015.

[24] Deepak Ghimire, Sunghwan Jeong, Joonwhoan Lee, and San Hyun Park. Facial expression recognition based on local region specific features and support vector machines. *Multimedia Tools and Applications*, 76(6):7803–7821, 2017.

[25] Deepak Ghimire and Joonwhoan Lee. Geometric feature-based facial expression recognition in image sequences using multi-class adaboost and support vector machines. *Sensors*, 13(6):7714–7734, 2013.

[26] Deepak Ghimire and Joonwhoan Lee. Extreme learning machine ensemble using bagging for facial expression recognition. *Journal of Information Processing Systems*, 10(3):443–458, 2014.

[27] Spyros Gidaris, Praveer Singh, and Nikos Komodakis. Unsupervised representation learning by predicting image rotations. *arXiv preprint arXiv:1803.07728*, 2018.

[28] Patrick O Glauner. Deep convolutional neural networks for smile recognition. *arXiv preprint arXiv:1508.06535*, 2015.

[29] Ivan Gogić, Martina Manhart, Igor S Pandžić, and Jörgen Ahlberg. Fast facial expression recognition using local binary features and shallow neural networks. *The visual computer*, 36(1):97–112, 2020.

[30] Ian J Goodfellow, Dumitru Erhan, Pierre Luc Carrier, Aaron Courville, Mehdi Mirza, Ben Hamner, Will Cukierski, Yichuan Tang, David Thaler, Dong-Hyun Lee, et al. Challenges in representation learning: A report on three machine learning contests. In *International conference on neural information processing*, pages 117–124. Springer, 2013.

[31] Jean-Bastien Grill, Florian Strub, Florent Altché, Corentin Tallec, Pierre Richemond, Elena Buchatskaya, Carl Doersch, Bernardo Avila Pires, Zhaohan Guo, Mohammad Gheshlaghi Azar, et al. Bootstrap your own latent-a new approach to self-supervised learning. *Advances in neural information processing systems*, 33:21271–21284, 2020.

[32] Lionel Gueguen, Alex Sergeev, Ben Kadlec, Rosanne Liu, and Jason Yosinski. Faster neural networks straight from jpeg. *Advances in Neural Information Processing Systems*, 31, 2018.

[33] Michael Gutmann and Aapo Hyvärinen. Noise-contrastive estimation: A new estimation principle for unnormalized statistical models. In Yee Whye Teh and Mike Titterington, editors, *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, volume 9 of *Proceedings of Machine Learning Research*, pages 297–304, Chia Laguna Resort, Sardinia, Italy, 13–15 May 2010. PMLR.

[34] Raia Hadsell, Sumit Chopra, and Yann LeCun. Dimensionality reduction by learning an invariant mapping. In *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06)*, volume 2, pages 1735–1742. IEEE, 2006.

[35] Jihun Hamm, Christian G Kohler, Ruben C Gur, and Ragini Verma. Automated facial action coding system for dynamic analysis of facial expressions in neuropsychiatric disorders. *Journal of neuroscience methods*, 200(2):237–256, 2011.

[36] Kaiming He, Haoqi Fan, Yuxin Wu, Saining Xie, and Ross Girshick. Momentum contrast for unsupervised visual representation learning. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 9729–9738, 2020.

[37] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[38] Steven Hickson, Nick Dufour, Avneesh Sud, Vivek Kwatra, and Irfan Essa. Eyemotion: Classifying facial expressions in vr using eye-tracking cameras. In *2019 IEEE winter conference on applications of computer vision (WACV)*, pages 1626–1635. IEEE, 2019.

[39] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4700–4708, 2017.

[40] Heechul Jung, Sihaeng Lee, Junho Yim, Sunjeong Park, and Junmo Kim. Joint fine-tuning in deep neural networks for facial expression recognition. In *Proceedings of the IEEE international conference on computer vision*, pages 2983–2991, 2015.

[41] Dae Hoe Kim, Wissam J Baddar, Jinhyeok Jang, and Yong Man Ro. Multi-objective based spatio-temporal feature representation learning robust to expression intensity variations for facial expression recognition. *IEEE Transactions on Affective Computing*, 10(2):223–236, 2017.

[42] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[43] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.

[44] Yann LeCun. The mnist database of handwritten digits. *http://yann. lecun. com/exdb/mnist/*, 1998.

[45] Jia Zheng Lim, James Mountstephens, and Jason Teo. Emotion recognition using eye-tracking: Taxonomy, review and current challenges. *Sensors*, 20(8), 2020.

[46] Patrick Lucey, Jeffrey F Cohn, Takeo Kanade, Jason Saragih, Zara Ambadar, and Iain Matthews. The extended cohn-kanade dataset (ck+): A complete dataset for action unit and emotion-specified expression. In *2010 ieee computer society conference on computer vision and pattern recognition-workshops*, pages 94–101. IEEE, 2010.

[47] Katsutoshi Masai, Yuta Sugiura, Masa Ogata, Kai Kunze, Masahiko Inami, and Maki Sugimoto. Facial expression recognition in daily life by embedded photo reflective sensors on smart eyewear. In *Proceedings of the 21st International Conference on Intelligent User Interfaces*, pages 317–326, 2016.

[48] Albert Mehrabian. Communication without words. In *Communication theory*, pages 193–200. Routledge, 2017.

[49] Ishan Misra and Laurens van der Maaten. Self-supervised learning of pretext-invariant representations. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 6707–6717, 2020.

[50] Jingping Nie, Yigong Hu, Yuanyuting Wang, Stephen Xia, and Xiaofan Jiang. Spiders: Low-cost wireless glasses for continuous in-situ bio-signal acquisition and emotion recognition. In *2020 IEEE/ACM Fifth International Conference on Internet-of-Things Design and Implementation (IoTDI)*, pages 27–39. IEEE, 2020.

[51] Mehdi Noroozi and Paolo Favaro. Unsupervised learning of visual representations by solving jigsaw puzzles. In *European conference on computer vision*, pages 69–84. Springer, 2016.

[52] Aaron van den Oord, Yazhe Li, and Oriol Vinyals. Representation learning with contrastive predictive coding. *arXiv preprint arXiv:1807.03748*, 2018.

[53] Christopher Pramerdorfer and Martin Kampel. Facial expression recognition using convolutional neural networks: state of the art. *arXiv preprint arXiv:1612.02903*, 2016.

[54] Rajakumari and S. Tamil Selvi. Hci and eye tracking : Emotion recognition using hidden markov model. 2015.

[55] K Ramamohan Rao and Ping Yip. *Discrete cosine transform: algorithms, advantages, applications*. Academic press, 2014.

[56] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 779–788, 2016.

[57] Evangelos Sariyanidi, Hatice Gunes, and Andrea Cavallaro. Automatic analysis of facial affect: A survey of registration, representation, and recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 37(6):1113–1133, 2015.

[58] Florian Schroff, Dmitry Kalenichenko, and James Philbin. Facenet: A unified embedding for face recognition and clustering. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 815–823, 2015.

[59] Caifeng Shan, Shaogang Gong, and Peter W. McOwan. Facial expression recognition based on local binary patterns: A comprehensive study. *Image Vision Comput.*, 27(6):803–816, may 2009.

[60] Kihyuk Sohn. Improved deep metric learning with multi-class n-pair loss objective. *Advances in neural information processing systems*, 29, 2016.

[61] Gilbert Strang. The discrete cosine transform. *SIAM review*, 41(1):135–147, 1999.

[62] Yonglong Tian, Dilip Krishnan, and Phillip Isola. Contrastive multiview coding. In *European conference on computer vision*, pages 776–794. Springer, 2020.

[63] Yonglong Tian, Chen Sun, Ben Poole, Dilip Krishnan, Cordelia Schmid, and Phillip Isola. What makes for good views for contrastive learning? *Advances in Neural Information Processing Systems*, 33:6827–6839, 2020.

[64] Robert Torfason, Fabian Mentzer, Eirikur Agustsson, Michael Tschannen, Radu Timofte, and Luc Van Gool. Towards image understanding from deep compression without decoding. *arXiv preprint arXiv:1803.06131*, 2018.

[65] Aaron Van den Oord, Nal Kalchbrenner, Lasse Espeholt, Oriol Vinyals, Alex Graves, et al. Conditional image generation with pixelcnn decoders. *Advances in neural information processing systems*, 29, 2016.

[66] Laurens Van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of machine learning research*, 9(11), 2008.

[67] Aaron Van Oord, Nal Kalchbrenner, and Koray Kavukcuoglu. Pixel recurrent neural networks. In *International conference on machine learning*, pages 1747–1756. PMLR, 2016.

[68] Pascal Vincent, Hugo Larochelle, Yoshua Bengio, and Pierre-Antoine Manzagol. Extracting and composing robust features with denoising autoencoders. In *Proceedings of the 25th international conference on Machine learning*, pages 1096–1103, 2008.

[69] Chao-Yuan Wu, Manzil Zaheer, Hexiang Hu, R Manmatha, Alexander J Smola, and Philipp Krähenbühl. Compressed video action recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 6026–6035, 2018.

[70] Zhirong Wu, Yuanjun Xiong, Stella X Yu, and Dahua Lin. Unsupervised feature learning via non-parametric instance discrimination. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3733–3742, 2018.

[71] Kai Xu, Minghai Qin, Fei Sun, Yuhao Wang, Yen-Kuang Chen, and Fengbo Ren. Learning in the frequency domain. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1740–1749, 2020.

[72] Zhilin Yang, Zihang Dai, Yiming Yang, Jaime Carbonell, Russ R Salakhutdinov, and Quoc V Le. Xlnet: Generalized autoregressive pretraining for language understanding. *Advances in neural information processing systems*, 32, 2019.

[73] Mang Ye, Xu Zhang, Pong C Yuen, and Shih-Fu Chang. Unsupervised embedding learning via invariant and spreading instance feature. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 6210–6219, 2019.

[74] Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson. How transferable are features in deep neural networks? In Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 27. Curran Associates, Inc., 2014.

[75] Milad Mohammad Taghi Zadeh, Maryam Imani, and Babak Majidi. Fast facial emotion recognition using convolutional neural networks and gabor filters. In *2019 5th Conference on Knowledge Based Engineering and Innovation (KBEI)*, pages 577–581. IEEE, 2019.

[76] Ce Zhan, Wanqing Li, Philip Ogunbona, and Farzad Safaei. A real-time facial expression recognition system for online games. *International Journal of Computer Games Technology*, 2008, 2008.

[77] Guoying Zhao and Matti Pietikäinen. Dynamic texture recognition using local binary patterns with an application to facial expressions. *IEEE transactions on pattern analysis and machine intelligence*, 29:915–28, 07 2007.

[78] Nanxuan Zhao, Zhirong Wu, Rynson WH Lau, and Stephen Lin. What makes instance discrimination good for transfer learning? *arXiv preprint arXiv:2006.06606*, 2020.