Tampere University

Liang Fang

PERSON RE-IDENTIFICATION WITH DEEP LEARNING

Information Technology and Communication Sciences (ITC)
Master of Science Thesis
November 2019

ABSTRACT

Liang Fang: Person Re-identification with Deep Learning
Master of Science Thesis
Tampere University
Machine Learning and Data Engineering
November 2019

In this work, we survey the state of the art of person re-identification and introduce the basics of the deep learning method for implementing this task. Moreover, we propose a new structure for this task.

The core content of our work is to optimize the model that is composed of a pre-trained network to distinguish images from different people with representative features. The experiment is implemented on three public person datasets and evaluated with evaluation metrics that are mean Average Precision (mAP) and Cumulative Matching Characteristic (CMC).

We take the BNNeck structure proposed by Luo *et al*. [25] as the baseline model. It adopts several tricks for the training, such as the mini-batch strategy of loading images, data augmentation for improving the model's robustness, dynamic learning rate, label-smoothing regularization, and the L_2 regularization to reach a remarkable performance. Inspired from that, we propose a novel structure named SplitReID that trains the model in separated feature embedding spaces with multiple losses, which outperforms the BNNeck structure and achieves competitive performance on three datasets. Additionally, the SplitReID structure holds the property of lightweight computation complexity that it requires fewer parameters for the training and inference compared to the BNNeck structure.

Person re-identification can be deployed without high-resolution images and fixed angle of pedestrians with the deep learning method to achieve outstanding performance. Therefore, it holds an immeasurable prospect in practical applications, especially for the security fields, even though there are still some challenges like occlusions to be overcome.

Keywords: deep learning, person re-identification, metric learning, multi-loss, SplitReID

CONTENTS

1 INTRODUCTION

Nowadays, with the rapid development of society, machine learning has been widely used in many fields of scientific research and every aspect of life with an explosion of information such as audio, video, and other formats of data. Moreover, deep learning is the most powerful and efficient method of machine learning due to the dramatic improvement of computing power. It has made significant achievements in the field of computer vision compared to traditional methodologies of machine learning.

As one of the most concerned fields of artificial intelligence, computer vision makes machines have the capability of perceiving and understanding images and even performs better than humans in some applications such as face recognition. So it is widely used in some industries: security, finance, and retail. However, there are still several challenges with computer vision, such as occlusions, poor resolution images, and bad illumination conditions. For example, detecting if an image contains objects becomes one of the most challenging tasks in computer vision, in which the task is named as object detection.

Object detection is one of the core topics in computer vision. It locates objects in an image with bounding boxes and classifies detected objects. Some traditional approaches such as Scale invariant feature transform (SIFT) [24], Histogram of oriented gradient (HOG) [4] and Deformable part model (DPM) [7] were widely used before the popularity of deep learning. After that, there is a significant improvement of performance in object detection with the appearance of neural network based on object detectors, such as Region Convolutional Neural Network (RCNN) [9], You Only Look Once (YOLO) [30] and Single Shot Multibox Detector (SSD) [23].

Person re-identification is the task that retrieves the same person from a gallery set according to the query person based on object detection. It has considerable potential in urban security or tracking system. However, it faces several challenges, including uncertain angles and orientations of a person captured by different cameras, faces are blurred due to the distance from people to cameras, the similar color of dressing, and occlusion of the object. Research of person re-identification starts from the 1990s with some traditional methods. They extract the single feature, *i.e.*, obtain the histogram of color or shape from a person image, and make identification with Bayes' Theorem. The accuracy has got a significant improvement with the deep learning method Convolutional Neural Network (CNN) in recent years.

In our implementation of the person re-identification, the model is trained and evaluated

on three public person datasets: Market-1501 [44], DukeMTMC-reID [32], MSMT17 [41] and aim to achieve a competitive performance compared to the state-of-the-art methods. The main part of work is that we propose a novel structure named SplitReID that performs competitively on the three datasets with a relatively low computational complexity.

This thesis is structured as follows. Chapter 2 introduces some theoretical backgrounds and concepts of machine learning and deep neural network firstly. The principle and alternative components of neural network training, validation methodologies, and evaluation metrics would be introduced in the rest of this chapter. Chapter 3 presents the background of three person datasets, and the implementation details of the BNNeck and our proposed SplitReID structure. Chapter 4 firstly discusses comparisons about the performance among our proposed method and some state-of-the-art approaches on three datasets. Then another comparison among different backbone models is also presented. Chapter 5 summarizes the highlights of our proposed SplitReID compared to others' work, and the work to improve the performance presently.

2 THEORY

This chapter introduces the background and concepts of artificial neural network and machine learning. Besides, alternatives of components of our model are presented in the rest.

2.1 Machine Learning

Machine learning is a branch of artificial intelligence. It aims to make computers have the capability of self-learning that machines can make identification or prediction by learning from large amounts of historical data with algorithms. There is a well-known definition of machine learning from Tom Mitchell: A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E [26].

Machine learning becomes popular again since the 1990s and is now more powerful than ever before. It has been applied in lots of fields with excellent performance, such as health care, autonomous vehicles, financial transactions, and internet security. Even in 2016, AlphaGo produced by Google DeepMind defeated one of the best go players, which is a milestone that machine overcame humans in the field that is considered as the most difficult to beat.

Machine learning works as analyzing rules from data and generating results according to different requirements. It mainly consists of two classes: supervised learning and unsupervised learning.

2.1.1 Supervised Learning

Supervised learning is a machine learning task that trains a model learned from labeled training data and make a prediction for new data. In supervised learning, each training example consists of an input object and an expected output value that the model would build up a relation between the input and corresponding output. N training examples $(\boldsymbol{x}_t, \boldsymbol{y}_t)$ are in the form of $\{(x_1, y_1), \ldots, (x_N, y_N)\}$. A function g would be found by supervised learning that the model makes predictions $\hat{\boldsymbol{y}}$ from test samples $\boldsymbol{x}$, which is denoted as $\hat{\boldsymbol{y}} = g(\boldsymbol{x})$.

In supervised learning, prediction results can be continuous values or discrete values. Thus, it is divided into regression and classification problems. Figure 2.1 shows an example about regression and classification:

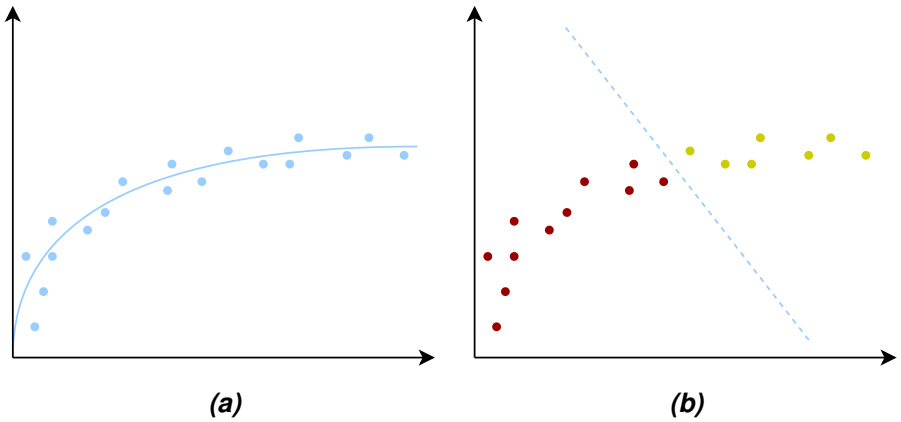


Figure 2.1. *Example of supervised learning. (a) Regression. The prediction value (blue curve) is continuous for fitting samples (blue dots). (b) Classification. It separates samples (red and yellow dots) into categories.*

- In the regression task, the goal is to predict the result with continuous output. For instance, a problem with predicting the house price when given the size of a house is a regression problem since the output price is a continuous output.
- Classification is the most common task in machine learning. It aims to find a decision boundary to predict results in the discrete domain, which maps the input into discrete categories. For example, the given data is about the medical data of cancer. It contains the size of the tumors and corresponding labels of "benign" or "malignant" of the tumor. The goal is to predict if the given tumor is benign or malignant, and this is a classification problem since the prediction results are discrete categories. The prediction $y_i \in \{1, \ldots, C\}$, where i is the number of samples and C is the number of classes.

2.1.2 Unsupervised Learning

Unsupervised learning refers to learn useful patterns from unlabelled data. Unsupervised learning algorithms generally learn directly from the original data without any manual operations such as labeling or feedback. The goal of unsupervised learning is to discover valuable information that hides in the data, including features, categories, structures, and distribution patterns.

Clustering is one of the most classic topics of unsupervised learning, and the goal is to divide a group of samples into different clusters according to specific criteria. A common rule is that samples in a group are more similar than those among groups. For example, consumption habit mining is a typical application of unsupervised learning, which clusters different groups of consumers from consuming features such as purchase frequency, and

amount. Consequently, specific marketing plans can be made according to different kinds of consumers. Figure 2.2 shows an example of unsupervised learning.

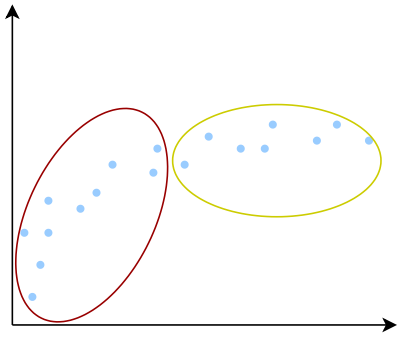


Figure 2.2. *Example of unsupervised learning. The algorithm clusters samples (blue) that are without labels into two groups (red and yellow circles) with a specific criteria.*

2.2 Deep Learning

Deep learning is a new field of machine learning research, which has brought machine learning closer to its original goal: artificial intelligence.

Deep learning extracts representative features that can be calculated by computers from various formats of data such as images or audios based on multi-layer neural network. It solves a core problem of machine learning that a complex feature could be combined or extracted from several simple features automatically.

Currently, there are two mainstream neural network structures, which have been proved that they are beneficial in some fields respectively: convolutional neural network is applied for spatially-distributed data such as images in image recognition and recursive neural network deals with time-distributed data such as audio signal in speech recognition and natural language processing.

2.2.1 Neural Network

The artificial neural network is a network, which consists of processing units (neurons), and it simplifies and simulates the biological neural network. The information processing of the neural network is achieved through communication among neurons. The performance of prediction depends on the dynamic optimization of weights.

Multiple neurons are connected in a network, one of which can receive various input signals and convert them into output signals according to specific rules. Because of the intricate connection and non-linearity of transmission signals among neurons, varieties of relations can be built from input to output. The artificial neural network is described as a black box that can give the expression of objective mapping between input and output.

Figure 2.3 shows the structure of an aritificial neuron. The input $(x_1, \ldots, x_N)$ and weight vector $(w_1, \ldots, w_N)$ are denoted as $\boldsymbol{x}$ and $\boldsymbol{w}$, respectively. The intermediate output z is obtained with a linear operation $\sum$ by inner product $\boldsymbol{w} \cdot \boldsymbol{x}$. Then the final output a is generated after passing z through an activation function σ. The goal of neural network training is to adjust weights to make the prediction as accurate as possible.

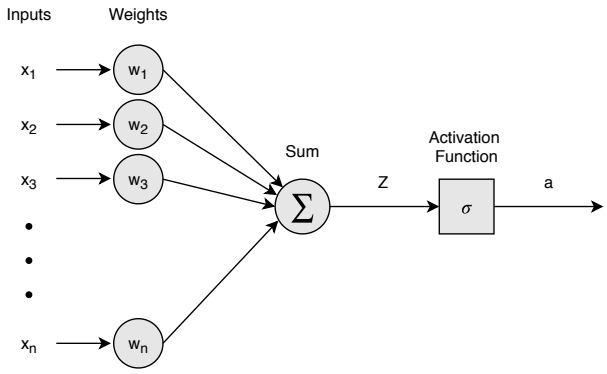


Figure 2.3. *Basic structure of a neural network. The input vector $\boldsymbol{x}$ is transmitted for the linear computation $\sum$ with weights $\boldsymbol{w}$, and the output a is obtained from the activation function σ with previous computation result z.*

An artificial neural network consists of multiple neurons, which is usually divided into three parts: the input layer, the hidden layer, and the output layer. Figure 2.4 is an illustration of the fully-connected neural network.

- The first layer is the input layer, which receives input vector and passes them into the neural network model.
- The last layer is the output layer, which generates the final result calculated from previous layers.
- Layers between the input layer and the output layer are hidden layers. Information is transmitted among neurons in hidden layers.

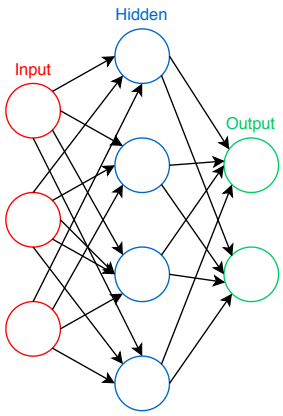
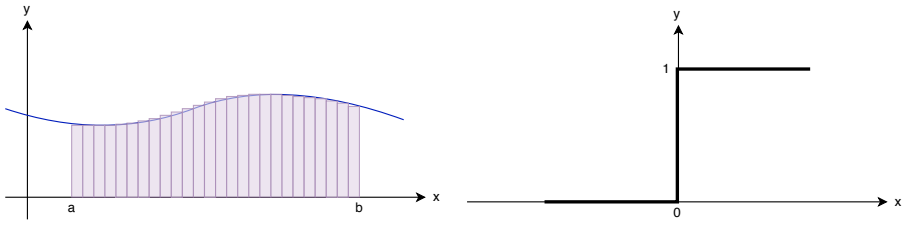


Figure 2.4. *Example of the fully-connected neural network. It contains one layer for the input layer, the hidden layer, and the output layer, respectively. Each neuron is connected to every neuron in the previous layer, from input to output. Besides, the number of hidden layers depends on different requirements.*

Activation Function

The activation function is to make the linear output to be non-linear, which enables the neural network model to fit any functions according to the Universal Approximation Theorem [14]. If the neural network is without the activation function, in which case the output is the linear combination of inputs no matter how many layers the network has, the final result equals to that generated from adequate neurons without hidden layers. The approximation capability of the network is limited.

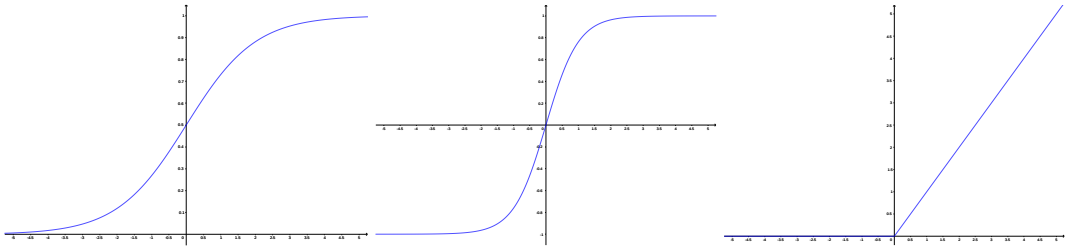
Similar to the basic idea of calculus: the area under the curve shown in Figure 2.5a can be divided into an infinite number of rectangles, and the area can be worked out easily by summing rectangles up. So, activation functions like Heaviside function shown in 2.5b are needed to “squeeze” the input to approximate curves.



(a) ***(b)***

Figure 2.5. *(a) Illustration of calculus. The area calculation under the curve (shadow) can be worked out with the integral of the infinite number of rectangles. (b) Heaviside function. The output equals* 1 *if the input is greater than* 0*, and* 0 *otherwise.*

There are three activation functions shown in Figure 2.6 are commonly used so far:



(a) ***(b)*** ***(c)***

Figure 2.6. *(a) Logistic Sigmoid Function. (b) Tanh Function. (c) ReLU [27] Function.*

- Figure 2.6a is the logistic sigmoid function. It maps the wide-range input into the interval $(0, 1)$. It is effective when the difference of outputs is not particularly large. It is usually applied in the last layer of a binary classification model to output the "probability" of each class. The logistic sigmoid function is defined as:

$$Logistic\ Sigmoid(x) = \frac{1}{1 + e^{-x}}. \tag{2.1}$$

- Figure 2.6b is the tanh function. It maps the input into interval $(-1, 1)$. It tackles a problem of logistic sigmoid that the function is not symmetric of the origin, which

makes convergence faster than logistic sigmoid. The tanh function is defined as:

$$Tanh(x) = \frac{2}{1 + e^{-2x}} - 1. \tag{2.2}$$

- Figure 2.6c is the ReLU [27] function. The output equals 0 if the input is less than 0, while the output equals the input otherwise. It solves the vanishing gradient problem that the gradient attenuates dramatically to approximately zero with the layer deepens, which makes the training stagnating. ReLU makes the convergence faster than the logistic sigmoid and tanh. The ReLU function is defined as:

$$ReLU(x) = \begin{cases} 0, & x < 0 \\ x, & x \geq 0 \end{cases}. \tag{2.3}$$

- Softmax is a special activation, which is applied in the output layer of classification task specifically. It is a extension of the logistic sigmoid, which is utilized in multi-class classification. The formula of softmax is defined as:

$$\sigma(\boldsymbol{z})_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}} \quad for\ j = 1, \ldots, K. \tag{2.4}$$

  The prediction result $\sigma(\boldsymbol{z})_j$ is the probability of j class $\in (1, \ldots, K)$. The softmax converts a K-dimensional vector $\boldsymbol{z}$ into another K-dimensional vector $\sigma(\boldsymbol{z})$ that each element is in the interval $(0, 1)$, and the sum of all elements equals 1.

2.2.2 Convolutional Neural Network

Fully-connected neural network is not proper in image processing. An image is composed of multiple pixels. If the fully-connected network is applied for the image task, the number of parameters would be exploded since neurons are fully connected between layers, and one iteration of back-propagation would be a tremendous calculation. Figure 2.7 shows an example of the fully-connected network.

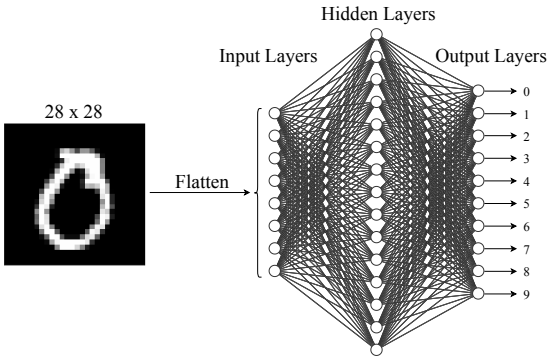


Figure 2.7. *Example of a fully-connected network with MNIST [20]. The Input layer contains 784 neurons flattened from a 28×28 image. The hidden and output layers are with 15 and 10 neurons, respectively. Lastly, this network is with 11,935 trainable parameters.*

Convolutional neural network (CNN) is particularly powerful in image recognition. It reduces the number of parameters massively by local connection and weight sharing in contrast to the fully-connected network. There are three regular components in the convolutional neural network, which are the convolutional layer, the pooling layer and the fully-connected layer that will be introduced in the following. Figure 2.8 illustrates a common structure of convolutional neural network.

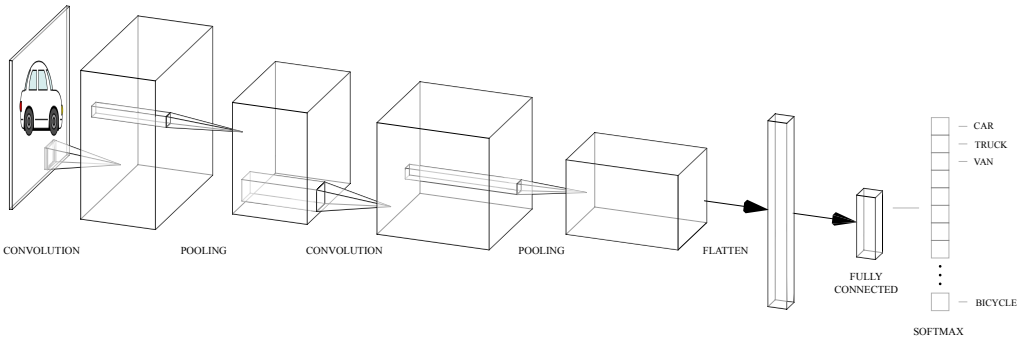


Figure 2.8. *Example of convolutional neural network. It makes a classification for the type of vehicles when given an image. The original image is passed through the convolutional layer, the pooling layer, and the fully-connected layer gradually with activation functions to a feature vector for the classification.*

Convolutional Layer

"Convolutional neural network" is named from the convolution operation. Image is a kind of spatial signal that the two-dimensional convolution operation is beneficial to retrieve the local feature from an image in the convolutional layer.

The convolution operation is to apply a mathematical operation for the original image with specific feature detectors that are called filters. The definition of original two-dimensional convolution is as blow, in which $f(i,j)$ and $g(i,j)$ are the original image and the filter, respectively. "$*$" symbolizes the convolution operation:

$$(f*g)(i,j) = \sum_{m=-\infty}^{m=+\infty} \sum_{n=-\infty}^{n=+\infty} f(m,n)g(i-m,j-n). \tag{2.5}$$

According to the commutative law of convolution, it can be expressed as:

$$(f*g)(i,j) = \sum_{m=-\infty}^{m=+\infty} \sum_{n=-\infty}^{n=+\infty} f(i-m,j-n)g(m,n). \tag{2.6}$$

In practice, the convolution operation in CNN is flipped from the original convolution that filters apply the dot product with the original image correspondingly. The definition is as:

$$(f*g)(i,j) = \sum_{m=-\infty}^{m=+\infty} \sum_{n=-\infty}^{n=+\infty} f(i+m,j+n)g(m,n). \tag{2.7}$$

Figure 2.9 shows an example of the convolution for a 7×7 image with a 3×3 filter.

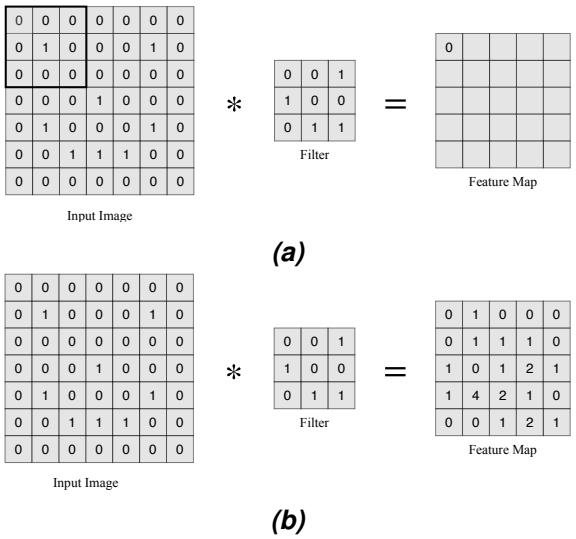


Figure 2.9. *Example of convolution operation in CNN. (a) First step of the convolution operation. It applies the filter for the first* 3×3 *window with the dot product and obtains* 0 *as the first element of the result. (b) Result of the convolution operation. The entire result is worked out, which is a* 5×5 *feature map.*

Furthermore, multiple feature maps could be obtained by applying different initialized filters for the original image. Figure 2.10 shows an example of multiple feature maps.

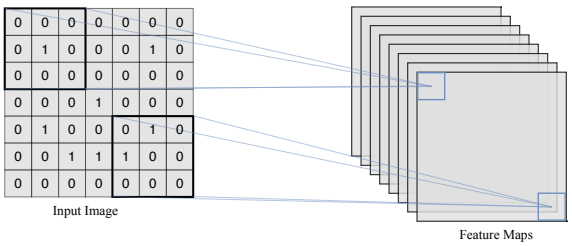


Figure 2.10. *Example of multiple feature maps. A set of feature maps are obtained with eight different filters, that the dimension of the feature maps is* $8 \times 5 \times 5$*.*

Besides, a colorful image usually contains RGB three channels. The two-dimensional filter is extended to be three-dimensional with the depth dimension and applied for the three channels simultaneously for generating feature maps. Figure 2.11 shows an example of the filter extended from 2D to 3D.

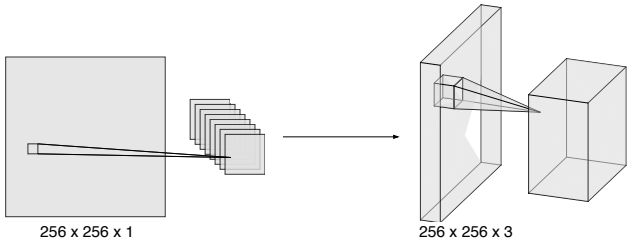


Figure 2.11. *Example of the filter extended from 2D to 3D. The filter is 2-dimensional when given a one channel image and it is extended automatically with the depth of the given image.*

Filter kernel is the core component for the feature retrieval, which are not specified with particular locations. So, the filter kernel can also be applied at arbitrary locations in an image, which shows the principle of “weight sharing” and reduces the number of trainable parameters effectively.

Pooling Layer

The pooling layer is usually after a convolutional layer, which is to continuously narrow the size in two-dimension to reduce the number of trainable parameters and computational complexity in the network. So, the pooling layer shortens the training time and avoids over-fitting.

Max pooling layer is the most commonly used pooling layer, which takes the maximum value in each window. Window size for pooling operation should be specified in advance, which determines the size of the output feature map. The advantage of the max pooling layer is that the output maintains even if the image shifts with several pixels. Hence, it is beneficial for reducing noise. Figure 2.12 shows an example of the max pooling operation for a 5×5 feature map with a 3×3 window.

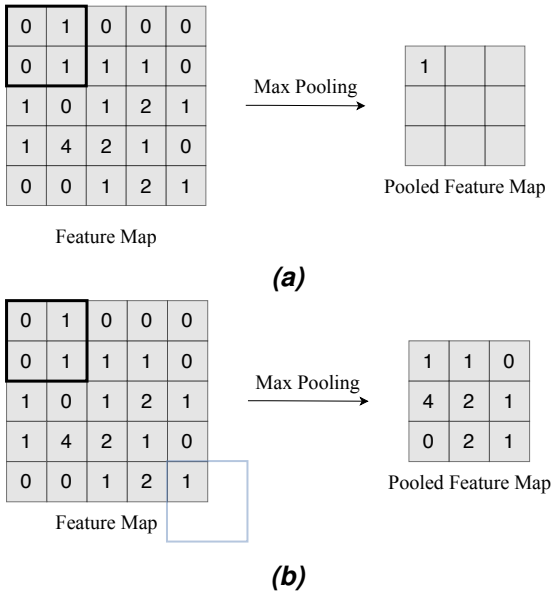


Figure 2.12. *Example of max pooling operation. (a) First step of the max pooling operation. It takes* 1 *that is the maximum value in the* 2×2 *window. (b) Result of the maxing pooling with zero padding. The entire result is worked out with the max pooling operation at two strides. Notably, zero-padding is applied in the margin of the feature map to guarantee the* 2×2 *max pooling window.*

The average pooling layer is also commonly used, which calculates the average value in a pooling window instead of the maximum number. It is useful for the situation that the variance of values in the window is large. However, the max pooling is verified as a better pooling method compared to average pooling through experiments in recent years.

Fully-Connected Layer

The fully-connected layer is to “summarize” the feature extracted from previous layers and map them into the sample space of categories. Therefore the fully-connected layer is adopted as the last few layers.

Besides, the flatten layer is usually appended before the fully connected layer to ensure the input shape to the fully-connected layer is a vector. Figure 2.13 shows an example of the flatten layer.

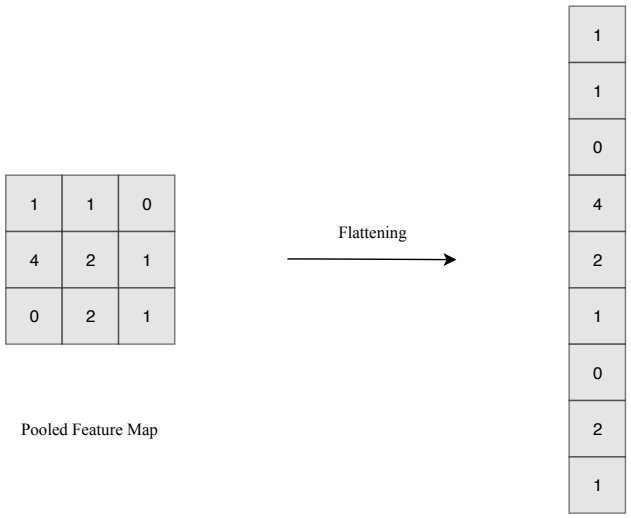


Figure 2.13. *Example of the flatten layer. It flattens a 3×3 feature map into a nine-dimension vector.*

Finally, the probability of each class is calculated after the fully-connected layer with activation functions. Figure 2.14 illustrates an example of the process for a classification task with a convolutional neural network.

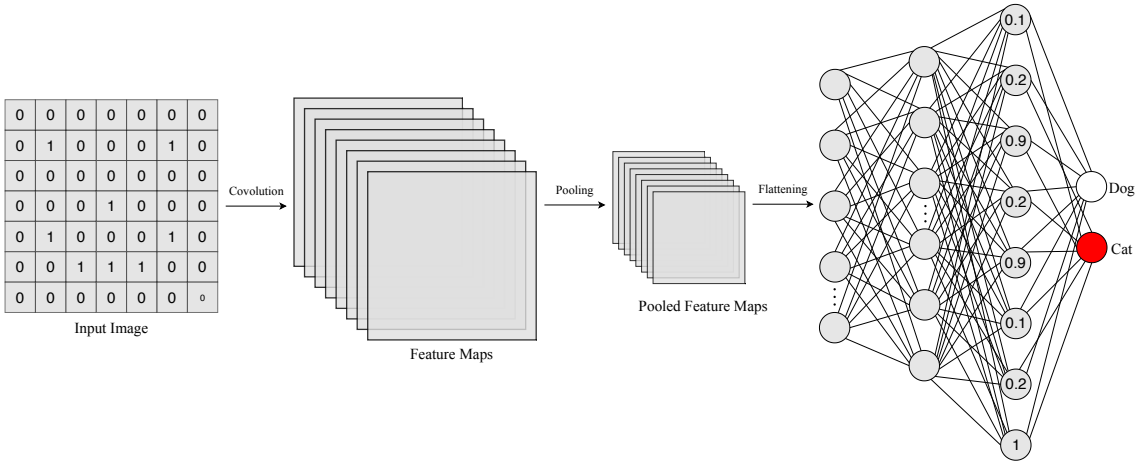


Figure 2.14. *Example of the process of a classification task. Given a 7×7 image, a set of feature maps are generated from the convolutional and pooling layers. Four fully-connected layers are adopted after the flatten layer to predict the animal species in the image with softmax.*

Two additional fully-connected layers are commonly set before the last fully connected layers since each fully-connected layer is expected to be responsible for different sub-features belong to categories. Figure 2.15 is an illustration of four fully-connected layers for classification..

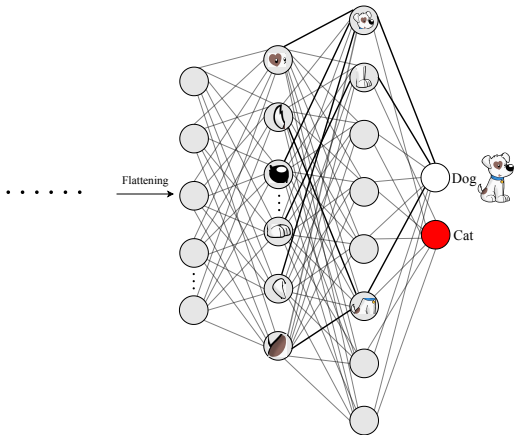


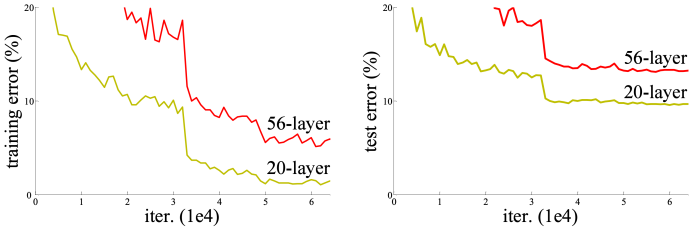
Figure 2.15. *Example of classification with three fully-connected layers. The last layer is to predict if the image is a dog or cat, and it requires the "local" features, such as face or leg, detected by the second last layer. Similarly, sub-features from limbs that the third last layer attempts to retrieve is for the next layer. A suitable number of fully-connected layers helps improve prediction accuracy effectively.*

2.3 Common Convolutional Network Structures

This section presents two widely used CNN structures: ResNet and DenseNet, which are implemented in our project.

2.3.1 ResNet

ResNet [12] is proposed for solving the degradation problem that both training and test error rate increases with the number of layers of a network deepens. Figure 2.16 shows an illustration of degradation problem. The performance of training and test becomes worse together with the model being deeper, which is not the over-fitting accordingly.



(a) ***(b)***

Figure 2.16. *Illustration of degradation problem. Both training (a) and test (b) error with 56 layers (red curve) is higher than that with 20 layers (yellow curve) on CIFAR-10. [12]*

Degradation problem implies that deep neural network is difficult to optimize when parameters in previous layers have been tuned properly so that the “identity mapping” that maps the input directly into the output without manipulation can avoid the degradation problem happen. ResNet introduces the “residual network” shown in Figure 2.17 as the basic structure to solve the problem.

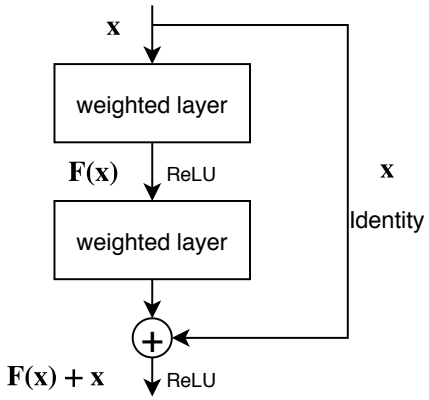


Figure 2.17. *Illustration of the residual network. It adds a shortcut from the identity $\boldsymbol{x}$ to the output $H(\boldsymbol{x}) = F(\boldsymbol{x}) + \boldsymbol{x}$ with manipulation $F(\boldsymbol{x})$. Furthermore, the residual $F(\boldsymbol{x}) = H(\boldsymbol{x}) - \boldsymbol{x}$ is the object that we expect to optimize instead of the whole result $H(\boldsymbol{x})$. In the case that the residual $F(\boldsymbol{x})$ will be optimized along with $H(\boldsymbol{x})$ if previous weights in $\boldsymbol{x}$ are not optimal, while $F(\boldsymbol{x})$ would be optimized into zero otherwise. [12]*

In practical, the architecture of residual networks varies depends on the depth of ResNet. Figure 2.18 shows two different residual networks.

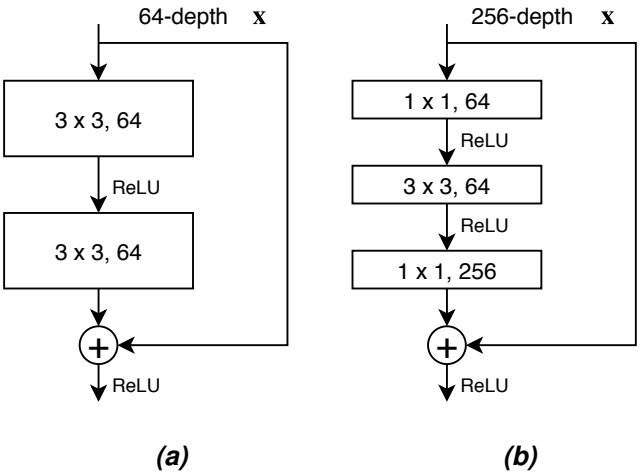


Figure 2.18. *Illustration of two different residual networks. (a) Building block of residual network for ResNet-34. It is for ResNet-34 that contains 34 layers of convolutional layer and fully-connected layer totally. Besides, it consists of two convolutional layers with 3×3 filters.(b) Residual network for deeper versions of ResNet. It is named as “Bottleneck” building block for ResNet-50/101/152. It contains two convolutional layers with 1×1 filters, which are for reducing and increasing the dimensionality of feature maps with the consideration of computational complexity. [12]*

2.3.2 DenseNet

DenseNet [15] is proposed for solving the vanishing-gradient problem that occurs with the increasing depth of networks. The main idea of the solution is to shorten the paths from the first layers to later layers that DenseNet connects all the layers to ensure the maximum information transmission between layers. Figure 2.19 illustrates the structure of DenseNet with a 5-layer dense block.

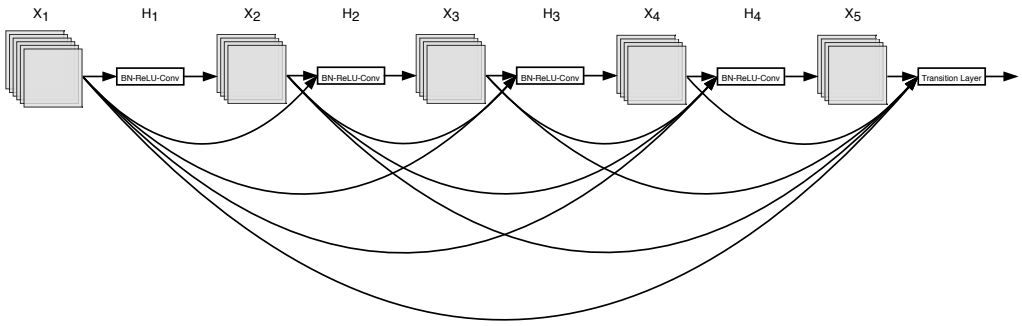
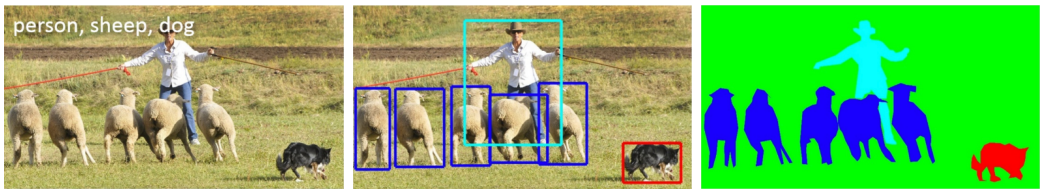


Figure 2.19. *Structure of DenseNet with a 5-layer dense block.* $X_i, i \in \{0, \ldots, 4\}$ *refers to the inputs for each layer and* $H_i, i \in \{1, \ldots, 4\}$ *indicates the operations of batch normalization, ReLU and convolution in layer. Each layer in the block is connected to previous ones to maximize the utilization of features processed beforehand. Besides, a transition layer that contains a convolutional layer and a pooling layer is added after each dense block to halve the tensor dimensionality. Assume there are* L *layers in the block, there will be* L *connections in traditional networks, while the number of connections in DenseNet is* $\frac{L(L+1)}{2}$ *.*

Dense block is the fundamental component and builds up a various depth of DenseNet depends on the requirement. DenseNet substantially reuses features and further alleviates the vanishing-gradient problem due to the dense-connected relation among layers. Besides, DenseNet requires a smaller number of parameters compared to ResNet.

2.4 Object Detection

Object detection is in the pipeline of the person re-identification task, where the feature extraction is based on the detected person images. It focuses on a specific object target and is the understanding of the foreground and background image, in which targets we are interested will be separated. In this section, two kinds of object detection algorithms are presented, which are region-based and regression-based algorithms: the former one contains two stages that are selecting regions of objects and identify for the detection. While the latter one combines those two procedures into one, which fastens the detection. Figure 2.20 shows three tasks of image understanding: classification, detection and segmentation.



(a) *(b)* *(c)*

Figure 2.20. *Three tasks of image understanding. (a) Classification. It divides objects in an image into categories. (b) Detection. It highlights objects in an image with a location that contains coordinates of bounding-boxes. (c) Segmentation. It segments the pixels of objects in an image. [21]*

2.4.1 Region-based Object Detection

The main idea of region-based object detection is to slide different sizes of windows in an image, then make classification and regression in each area of windows. Windows are usually set to fit one object with various sizes, and then every object can be detected by sliding ideally.

R-CNN

R-CNN [9] firstly selects region proposals with external generators such as selective search [39] that utilizes the hierarchical grouping algorithm to cluster tiny segments into region proposals with diverse properties of images. Secondly, it adopts the CNN model to extract features for the object classification and localization. However, the shortcoming of R-CNN is not efficient since each region proposal needs to be processed independently. Figure 2.21 is an illustration of how R-CNN works.

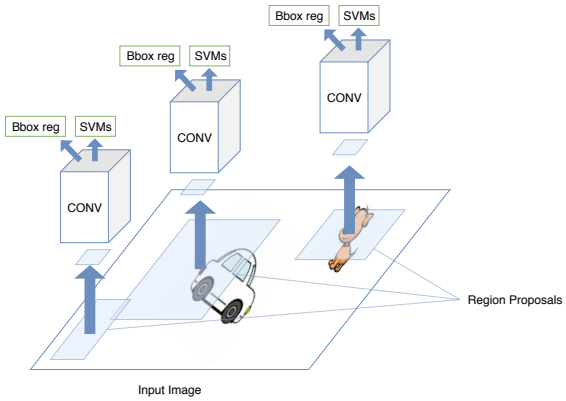


Figure 2.21. *Process of R-CNN. Firstly, about two thousand region proposals are selected by the selective-search method. Secondly, the CNN model is utilized to retrieve the feature. Finally, classification and localization are made from those region proposals with SVM [3] and linear regression method.*

Fast R-CNN

In fast R-CNN [8], The feature extraction is implemented by CNN for once from the original image in contrast to R-CNN. The bottleneck of the fast R-CNN is that the speed of region proposals generated from the original image is not fast enough. Figure 2.22a is an illustration of how fast R-CNN works: firstly, the general feature of the original image is extracted for sharing. Secondly, multiple region proposals generated from the original image retrieve their corresponding features from the general feature. Thirdly, the Region of Interest pooling method (RoI pooling) is applied to unify the feature dimension. Finally, the classification and localization are made with fully-connected layers, in which the classification is from the softmax, and the bounding box regression is from the fully-connected layer directly.

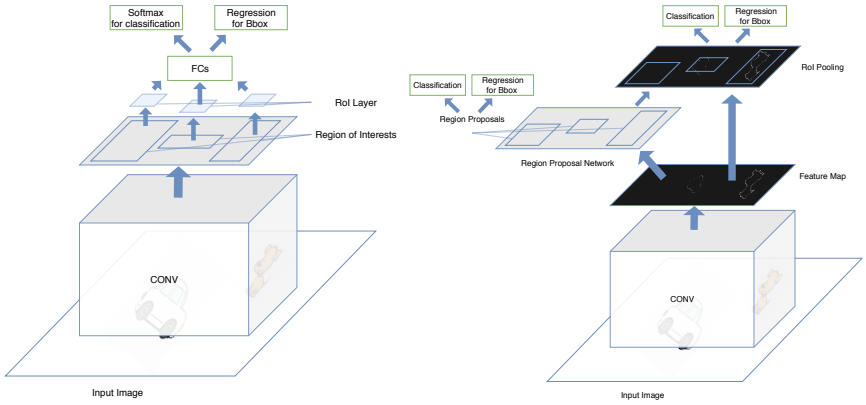


Figure 2.22. *(a) Process of Fast R-CNN. (b) Process of Faster R-CNN.*

Faster R-CNN

Faster R-CNN [31] proposes the Region proposal Networks (RPN) as the region proposal selector instead of the selective-search method. It integrates the feature extraction, region proposal selection, bounding box regression, and classification into a network, which accelerates the detection speed dramatically. Figure 2.22b is an illustration of how faster R-CNN works: firstly, the general feature map is extracted from the original image, which is the same as Fast R-CNN. Secondly, the corresponding receptive fields in the original image from each pixel of the general feature map is set for nine anchor boxes with different sizes. Thirdly, the Region Proposal Networks selects those anchors that contain an object as the region proposals. Finally, the chosen region proposals are with the corresponding feature from the general feature map to be made classification and regression after the RoI pooling.

2.4.2 Regression-based Object Detection

Region-based object detection includes two stages that are region proposal selection and classification-regression, which makes the speed of detection is not fast enough even though it performs well. Regression-based object detection makes classification and regression directly, which is usually more efficient and can achieve real-time performance.

You Only Look Once

You Only Look Once (YOLO) [30] divides a image into $S \times S$ grids, and predicts B bounding boxes in a grid and C classes for each grid. For each grid cell, coordinates of B bounding boxes, the confidence score of containing an object for B bounding boxes $Pr(Object) * IOU^{truth}_{predict}$ and the probability of classes for each grid $Pr(Class_i|Object), i \in \{1, \ldots, C\}$ are predicted based on GoogleNet [37], then the confidence score of each class for bounding boxes $Pr(Class_i) * IOU^{truth}_{predict}$ can be calculated by the conditional probability:

$$Pr(Class_i|Object) * Pr(Object) * IOU^{truth}_{predict} = Pr(Class_i) * IOU^{truth}_{predict}. \tag{2.8}$$

Furthermore, the output feature dimension for an image is $S \times S \times (B \times 5 + C)$, in which 5 refers to the four coordinates and one confidence score of containing an object. The final bounding box of each object is optimized with the Non-Maximum Suppression (NMS) [28]. Figure 2.23 is an example of YOLO.

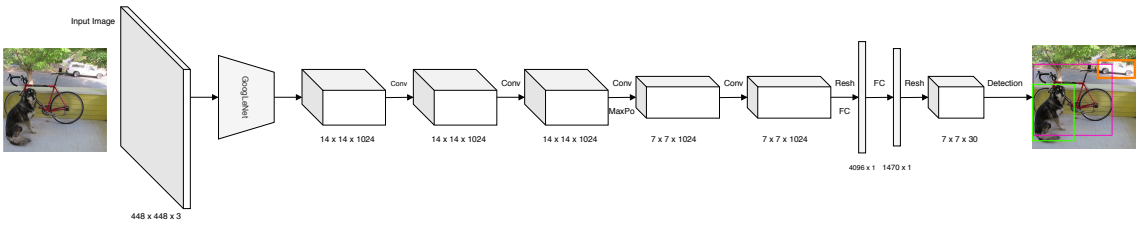


Figure 2.23. *Example of YOLO. In this example, $S, B,$ and C are set at $7, 2,$ and 20, respectively. So the out tensor is $7 \times 7 \times 30$ from a 448×448 image through a pre-trained CNN model. Finally, the dog, car, and bicycle are detected with the optimization of NMS.*

Single Shot Multibox Detector

Single Shot Multibox Detector (SSD) [23] is an improvement of YOLO:

- It adopts a "Pyramidal Feature Hierarchy" that yields feature maps with different sizes for detection, which can fit various images.
- It uses convolutional layers for the classification and regression instead of the last two fully-connected layers in YOLO.

- More anchor boxes with different sizes are designed in a grid cell. The dimension of the output tensor is $(C + 4) \times k \times m \times n$, in which C is the number of classes, k is the number of anchor boxes in a cell and $m \times n$ is the number of grid cells in the original image. Figure 2.24 is an example of SSD.

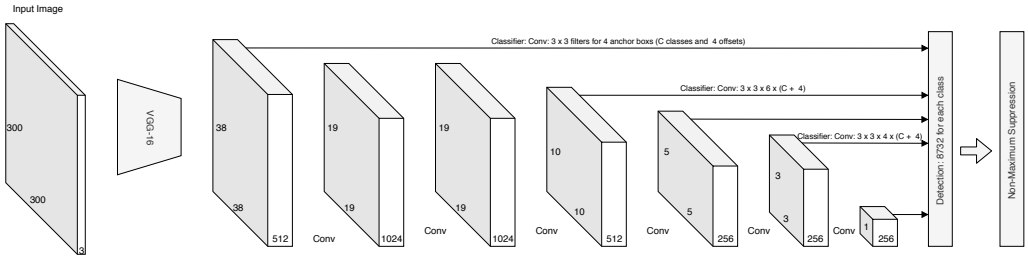


Figure 2.24. *Example of SSD. Passed through a VGG [34] model, feature maps with different sizes are generated with multiple convolutional layers from a* 300×300 *image for detection. Finally, objects are detected by optimizing with the NMS from 8732 bounding boxes for each class. Notably, feature maps with different sizes are processed with different number of anchor boxes for each grid cell. [23]*

2.5 Network Training

This section introduces essential components, including the learning rate, optimization algorithm, and loss function for the neural network training. Besides, the basic principles of the forward-propagation and back-propagation algorithms are presented.

2.5.1 Loss Function

The loss function is for measuring the inconsistency between the prediction of the machine learning model and the ground truth. It is a non-negative real value function, which is usually denoted as $l(y, h_\theta(x))$. The loss function is the objective function to be optimized in the training procedure. The smaller the loss is, the better the model performs.

Given a training set $(\boldsymbol{x}, \boldsymbol{y}) = \{(x_1, y_1), \ldots, (x_N, y_N)\}$ with N samples, in which y_i is the ground truth. $\hat{y_i} = h_\theta(x_i)$ is the prediction from a prediction model h_θ. The loss is defined as:

$$L(\boldsymbol{x}, \boldsymbol{y}) = \frac{1}{N} \sum_{i=1}^{N} l(y_i, \hat{y_i}). \tag{2.9}$$

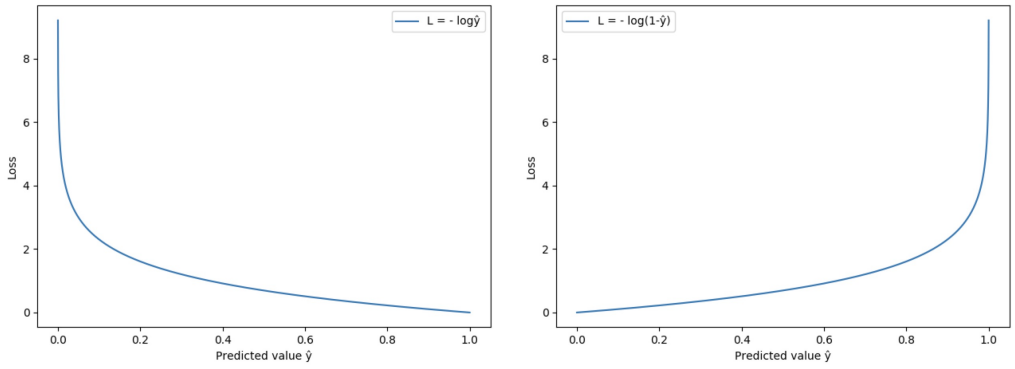
Cross Entropy Loss

Cross entropy loss is the most common loss function for classification. It describes the distance between two probability distribution, the smaller the cross entropy is, the closer

they get to each other. The basic format of cross entropy loss, which is the binary classification as below:

$$L = -y \log \hat{y} - (1 - y) \log(1 - \hat{y}). \tag{2.10}$$

Figure 2.25 illustrates the relation between ground truth and prediction in cross entropy loss. With the increasing difference between the prediction and ground truth, the "penalty" for the loss becomes more serious. Consequently, the model is forced to make the prediction closer to the ground truth.



(a) ***(b)***

Figure 2.25. *Illustration of the cross-entropy loss. (a) The ground truth y equals 1. The cross entropy loss function is $L = -\log(\hat{y})$, the closer the prediction $\hat{y}$ is to the ground truth y, the smaller the loss L will be. (b) The ground truth y equals 0. The cross entropy loss function is $L = -\log(1 - \hat{y})$, the further the prediction $\hat{y}$ is to the ground truth y, the smaller the loss L will be.*

For a binary classification task, the cross entropy loss with N samples is as:

$$L = -\frac{1}{N} \sum_{i=1}^{N} y_i \log \hat{y}_i + (1 - y_i) \log (1 - \hat{y}_i). \tag{2.11}$$

For a multi-label classification task that each sample can be with multiple labels, and each label is regarded as an independent binary label, so the cross entropy loss of multi-label classification with N samples and C labels is as:

$$L = -\frac{1}{N} \sum_{i=1}^{N} \sum_{j=1}^{C} y_{ij} \log \hat{y}_{ij} + (1 - y_{ij}) \log (1 - \hat{y}_{ij}). \tag{2.12}$$

For a multi-class classification that each sample is with one label from several classes, so the cross entropy loss of multi-class classification with N samples and C labels is as:

$$L = -\frac{1}{N} \sum_{i=1}^{N} \sum_{j=1}^{C} y_{ij} \log \hat{y}_{ij}. \tag{2.13}$$

Contrastive Loss

Contrastive loss [11] is a metric learning loss for pair-wise samples. Given $(\vec{X_1}, \vec{X_2})$ a pair of input samples. Y is the label for this pair, which equals 0 if they are deemd similar, vice versa. D_W stands for $D_W(\vec{X_1}, \vec{X_2})$ that refers to the euclidean distance between feature embeddings $G_W(\vec{X_1})$ and $G_W(\vec{X_2})$, where G_W is the feature retrieval function:

$$D_W(\vec{X_1}, \vec{X_2}) = \|G_W(\vec{X_1}) - G_W(\vec{X_2})\|_2. \tag{2.14}$$

The general form of contrastive loss for P pairs of samples is defined as:

$$\begin{aligned} \mathcal{L}(W) &= \sum_{i=1}^{P} L(W, (Y, \vec{X_1}, \vec{X_2})^i), \\ L(W, (Y, \vec{X_1}, \vec{X_2})^i) &= (1-Y)L_S(D_W^i) + YL_D(D_W^i), \end{aligned} \tag{2.15}$$

where L_S and L_D are the partial loss functions for pairs of similar and dissimilar samples and required to be specified. The contrastive loss is practically defined in the form as below, in which m is the hyper-parameter of margin:

$$L(W, (Y, \vec{X_1}, \vec{X_2})) = (1-Y)\frac{1}{2}(D_W)^2 + Y\frac{1}{2}\max\{0, m - D_W\}^2. \tag{2.16}$$

Figure 2.26 is an illustration of the contrastive loss about the relation between the loss L and the distance D_W.

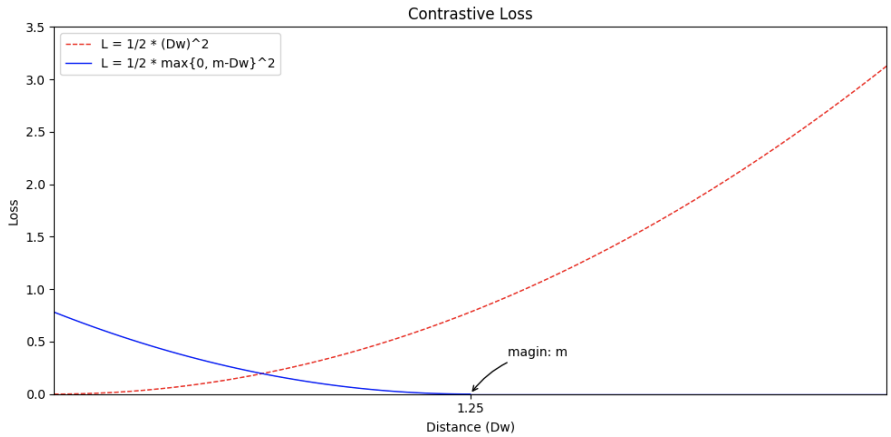


Figure 2.26. *Relation between the loss and the distance of a similar pair (red line) and dissimilar pair (blue line). $Y = 0$ represents that the pair-wise samples are similar and the loss function is $L = \frac{1}{2}(D_W)^2$. The loss gets larger with the increasing of the distance. While, $Y = 1$ represents that the pair-wise samples are dissimilar, and the loss function is $L = \frac{1}{2}\max\{0, m - D_W\}^2$. If the distance of the dissimilar pair is small, the loss increases to force the model to distinguish the dissimilar hard pair. Margin m is the threshold that tolerates the distance of a dissimilar pair.[11]*

Triplet Loss

Triplet loss [33] is initially proposed for face recognition, and it is proved that it performs well for the person re-identification. In face recognition, the motivation of triplet loss is to make the faces from the same person as close as possible and different people as far as possible in the embedding space.

Figure 2.27 shows an illustration of triplet loss. The input is a triplet that consists of Anchor, Positive, Negative:

- Anchor is the sample selected from the training set randomly.
- Positive is the randomly selected sample of the same class as anchor.
- Negative is the randomly selected sample that is different from anchor.

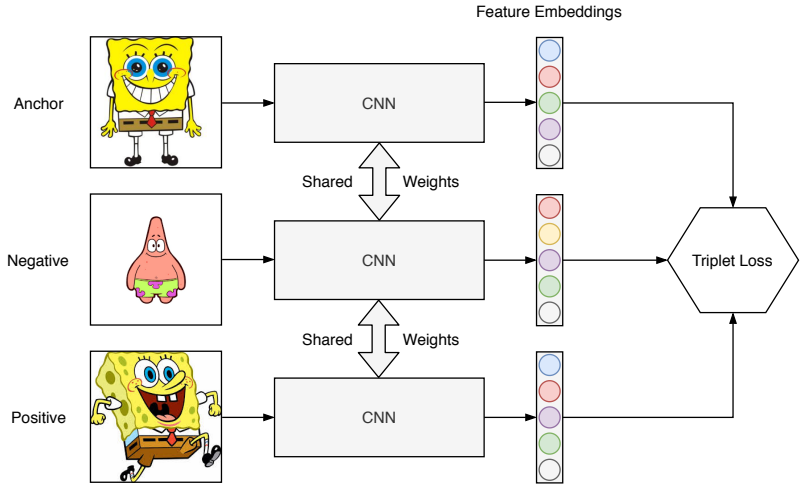


Figure 2.27. *Illustration of triplet loss. A triplet of anchor (SpongeBob), positive (SpongeBob), and negative (Patrick Star) images is as the input for extracting feature embeddings from the CNN model, and triplet loss is calculated from the feature embeddings of the triplet.*

The motivation of triplet loss is to make the embeddings of anchor and positive closer, while those of anchor and negative further. Figure 2.28 shows an example of the principle of triplet loss.

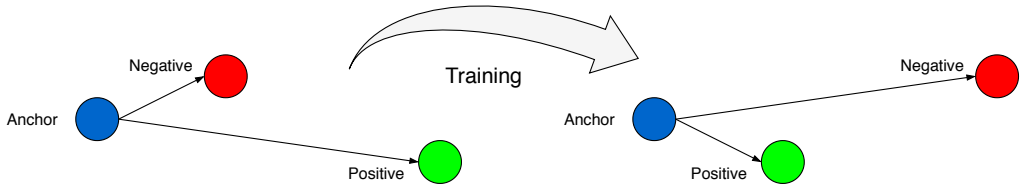


Figure 2.28. *Principle of triplet loss. The negative (red) is close to the anchor (blue), while the positive (green) is far away from the anchor before training, and their distances are optimized after training. [33]*

The definition of triplet loss is as below. (a, p, n) symbolizes the triplet of anchor, positive, negative. The notation d refers to the distance between two feature embeddings of samples, and $margin$ is the hyper-parameter that determines the threshold of satisfying $d(a, p) + margin < d(a, n)$:

$$L = \max\left(d(a, p) - d(a, n) + margin, 0\right). \tag{2.17}$$

The ideal situation of the triplet loss after training is that the distance $d(a, p)$ is close, and the $d(a, n)$ is far, which implies that the model could distinguish positives and negatives easily without a tolerance of margin, and the loss is 0. Otherwise, the loss is greater than 0. Generally, triplet loss is optimized with three situations of the triplet introduced below and illustrated in Figure 2.29:

- Easy negatives: triplet loss L is 0, that is, distance $d(a, p)$ is close and $d(a, n)$ is far naturally to satisfy $d(a, p) + margin < d(a, n)$.
- Hard negatives: $d(a, n) < d(a, p)$, which implies that the distance between the anchor and positive is further than that between anchor and negative.
- Semi-hard negatives: $d(a, p) < d(a, n) < d(a, p) + margin$, which implies that the distance between anchor and negative is "$margin$" further than that between anchor and positive.

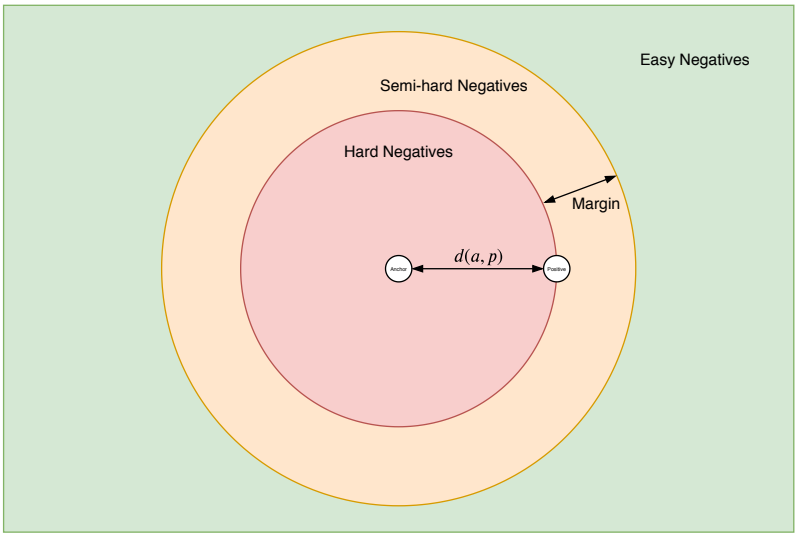


Figure 2.29. *Three situations of triplet loss optimization. The distance $d(a, p)$ is fixed at the radius of the red circle. The situation that the negative is within the red circle matches with "hard negatives". The negative sandwiched in the orange ring with $margin$ diameter corresponds to the "semi-hard negatives". The negative in the green area is for the "easy negatives".*

Lifted Structured Loss

Lifted structured loss [29] is proposed based on contrastive loss and triplet loss, which do not take full advantage of training batches: contrastive loss takes samples randomly as similar or dissimilar pairs, and the original triplet loss mines triplets of anchor, positive and negative randomly to form a training batch. However, the pair or triplet lacks a connection with each other. Lifted structured loss considers the relation between two arbitrary samples in the training batch. Figure 2.30 is an illustration of the relation among samples for three loss functions.

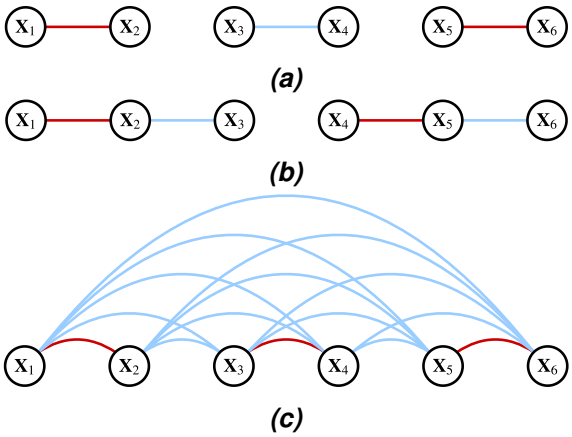


Figure 2.30. *Illustration of the relation among samples for three loss functions. Red and blue line denote the positive and negative pair, respectively. (a) Contrastive loss. Loss calculation is only based on pairs of* $(X1, X2)$, $(X3, X4)$ *and* $(X5, X6)$ *without* $(X2, X3)$ *or* $(X4, X5)$. *(b) Triplet loss. No relation is among triplets. (c) Lifted structured loss. Every two samples form into a pair. [29]*

The original lifted structured loss is defined as below, where $\widehat{\mathcal{P}}$ and $\widehat{\mathcal{N}}$ are the sets of positive and negative pairs, respectively in the whole training set. (i, j) indicates a positive pair, and (i, k) and (j, l) represent their neighbor negative pairs. D refers to the distance between two samples, and $\max(\max_{(i,k)\in\widehat{\mathcal{N}}} \alpha - D_{i,k}, \max_{(j,l)\in\widehat{\mathcal{N}}} \alpha - D_{j,l})$ illustrates that the harder negative pair is selected, in which α is the hyper-parameter of margin.

$$
\begin{aligned}
J &= \frac{1}{2|\widehat{\mathcal{P}}|} \sum_{(i,j)\in\widehat{\mathcal{P}}} \max(0, J_{i,j})^2, \\
J_{i,j} &= \max(\max_{(i,k)\in\widehat{\mathcal{N}}} \alpha - D_{i,k}, \max_{(j,l)\in\widehat{\mathcal{N}}} \alpha - D_{j,l}) + D_{i,j}.
\end{aligned}
\tag{2.18}
$$

However, there are two challenges for the implementation of the original lifted structured loss:

- It is a non-smooth loss function since it utilizes several hinge function $\max(0, \cdot)$, which is not beneficial for the gradient calculation.
- A lot of redundant calculation: the hardest negative pairs requires to be mined

repeatedly with iterating.

The first challenge is addressed by optimizing the original function into a smooth upper-bound version defined as below, which replaces the hinge operation with the sum operation:

$$\begin{aligned} \tilde{J} &= \frac{1}{2|\mathcal{P}|} \sum_{(i,j)\in\mathcal{P}} \max\left(0, \tilde{J}_{i,j}\right)^2, \\ \tilde{J}_{i,j} &= \log\left(\sum_{(i,k)\in\mathcal{N}} \exp\left\{\alpha - D_{i,k}\right\} + \sum_{(j,l)\in\mathcal{N}} \exp\left\{\alpha - D_{j,l}\right\} \right) + D_{i,j}. \end{aligned} \tag{2.19}$$

The second challenge is resolved by dividing the training set into several mini-batches with the stochastic approach, in which the mini-batch division is not random: a few positive pairs are picked randomly into a mini-batch alongside with their difficult negative neighbors. It guarantees that the mini-batch contains appropriate combinations of positive and negative pairs. Then, positive samples respectively compare their difficult negative neighbors to extract the hardest one. Figure 2.31 illustrates the hardest negative neighbors mining for a positive pair.

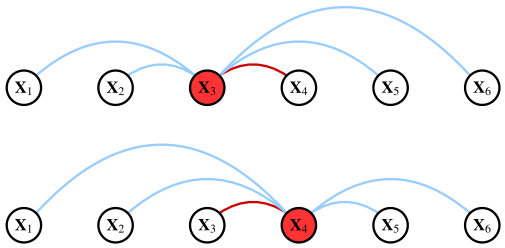


Figure 2.31. *Illustration of the hardest negative neighbors mining. $X3$ and $X4$ are the samples of a positive pair and they mine the hardest negative neighbors independently from $X1$, $X2$, $X5$, $X6$. [29]*

Arcface Loss

Arcface loss [6] is a state of the art loss function for face recognition. It maximizes the classification boundary in angular space based on the softmax function. The goal is to achieve the intra-class compactness and inter-class discrepancy. The motivation of the arcface loss is to combine the advantages of triplet loss and softmax loss that can:

- operate extensive comparison between images and classes. Softmax transfers the feature embedding into “scores” of each class with the fully connected layer and into the probability with the softmax function.
- have the hyper-parameter “margin” to control the balance between training difficulty and performance.
- support easy training with massive classes.

The process of forming the arface loss is introduced as below and Figure 2.32 illustrates the principle of the arcface loss:

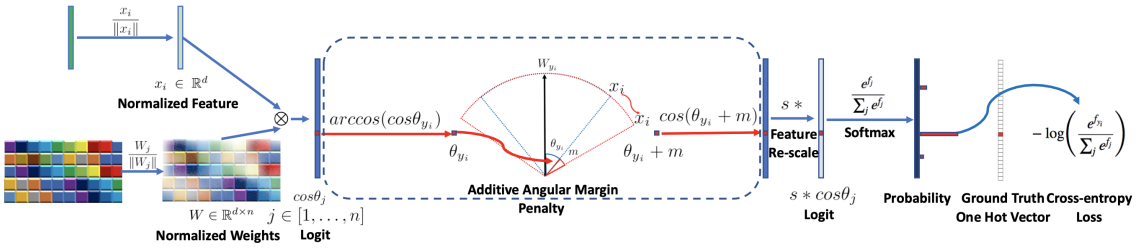


Figure 2.32. *Illustration of arcface loss. [6]*

1. Normalize both the feature vector x_i retrieved by CNN model and the weight matrix $W_j, j \in \{1, ..., n\}$ for each class. Then, the linear operation $W_j^T x_i = \|W_j\| \|x_i\| \cos\theta_j$ equals $W_j^T x_i = \cos\theta_j$ since $\|W_j\|$ and $\|x_i\|$ are unit vectors, where $\cos\theta_j$ is the vector of the cosine between x_i and each class.
2. Extract the target logit $\cos\theta_{y_i}$ that is the ground truth index in $\cos\theta_j$, and calculate the angle $\arccos(\cos\theta_{y_i})$ between the feature vector x_i and weight vector W_{y_i} of corresponding class.
3. Add the margin m for the angle between x_i and W_{y_i}, which controls the balance the performance and training efficiency. Re-scale the cosine with new angle as $s(\cos(\theta_{y_i} + m))$. Besides, margin m usually ranges from 2 to 3 degree.
4. Finally, obtain the arcface loss by applying the softmax function and cross entropy loss for outputs, in which N is the number of samples:

$$L = -\frac{1}{N}\sum_{i=1}^{N} \log \frac{e^{s(\cos(\theta_{y_i}+m))}}{e^{s(\cos(\theta_{y_i}+m))} + \sum_{j=1, j\neq y_i}^{n} e^{s\cos\theta_j}}. \tag{2.20}$$

Arcface loss adopts the "margin" to enforce the angle between target weight vector W_j and the feature embedding of the sample x_i to be small, and that between different target weight vectors to be large. Thus, the goal of intra-class compactness and inter-class discrepancy is achieved. Figure 2.33 is an illustration of the performance of arcface loss.

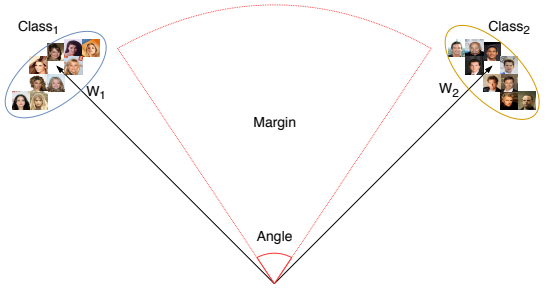


Figure 2.33. *Illustration of the performance of arcface loss. After training with arcface loss, samples in class* 1 *and* 2 *are compact with their targets* W_1 *and* W_2*, respectively. On the other hand,* W_1 *and* W_2 *are dispersed with the margin. [6]*

Ranked List Loss

Ranked list loss [40] is proposed for solving two problems in other metric learning loss function:

- One is that negative samples are not entirely utilized since most of them consider one or two negatives, such as a triplet or pair-wise samples.
- The other is that the intra-class structure is not maintained after training. The model is trained with a unit of samples instead of a cluster of the class. It makes samples from one class compact to a point.

Ranked list loss takes all the positive and negative samples with respect to an anchor sample into consideration, which "pushes" all negatives away from the anchor and "pulls" all positives over simultaneously. Figure 2.34 is an illustration of ranked list loss.

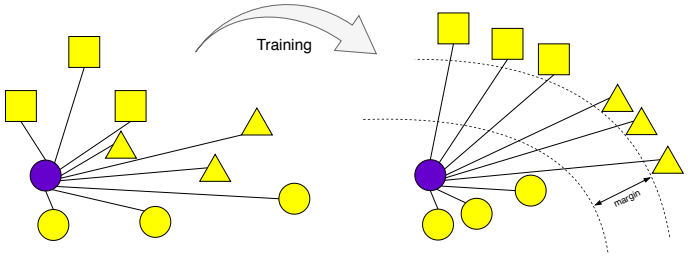


Figure 2.34. *Illustration of ranked list loss. Positives (yellow circle) and negatives (yellow squares and triangles) are closer and further to the anchor (purple circle) after trained with ranked list loss. Due to together movement for the positives and negatives, the intra-class structures are maintained. [40]*

Some notations for the introduction of ranked list loss: $\mathbf{X} = \{(\mathbf{x}_i, y_i)\}_{i=1}^{N}$ denotes as a batch of samples that includes N samples, where $(x_i, y_i), y_i \in \{1, 2, \ldots, C\}$ is the input and ground truth from C classes. $\{\mathbf{x}_i^c\}_{i=1}^{N_c}$ represents that all samples that are from class c in the batch and N_c is the number of these samples.

Given a pair of images $(\mathbf{x}_i, \mathbf{x}_j)$, the distance between negatives is expected to be higher than the threshold α, while the distance between positives is smaller than $\alpha - m$. Figure 2.35 is an illustration of the performance after training with ranked list loss.

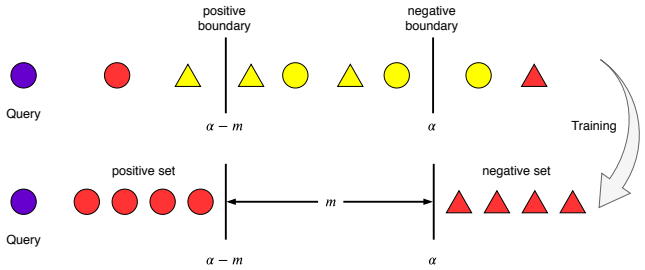


Figure 2.35. *Illustration of the performance with ranked list loss. After training with ranked list loss, negatives (square and triangle) are further than positives (circle) at margin m. Besides, the distance between two arbitrary positives is within a hyper-sphere with the radius at $\alpha - m$. [40]*

So the pairwise margin loss [40] is defined as below, in which $y_{ij} = 1$ if the pair is positive, otherwise $y_{ij} = 0$. f is the feature retrieval function from an image, and d is the distance of two feature embeddings:

$$L_m(\mathbf{x}_i; \mathbf{x}_j; f) = (1 - y_{ij})[\alpha - d_{ij}]_+ + y_{ij}[d_{ij} - (\alpha - m)]_+. \tag{2.21}$$

When given an anchor sample x_i^c that from class c, other samples in the batch can be ranked according to the similarity with x_i^c. $N_c - 1$ positives except the anchor in the batch is denoted as $P_{c,i} = \{\mathbf{x}_j^c | j \neq i\}$, and N_k negatives is denoted as $N_{c,i} = \{\mathbf{x}_j^k | k \neq c\}$. For positives, the loss function is defined as:

$$L_p(\mathbf{x}_i^c; f) = \frac{1}{\left|P_{c,i}^*\right|} \sum_{\mathbf{x}_j^c \in P_{c,i}^*} L_m(\mathbf{x}_i^c, \mathbf{x}_j^c; f). \tag{2.22}$$

Usually, the number of negatives is large, and the range of loss value is also large. Wang*et al*. [40] applies a weight w_{ij} for the basic negative loss to balance the performance, in which w_{ij} is defined as:

$$w_{ij} = \exp\left(T \cdot (\alpha - d_{ij})\right), \mathbf{x}_j^k \in N_{c,i}^*. \tag{2.23}$$

For negatives, the loss function is defined as:

$$L_N(\mathbf{x}_i^c; f) = \sum_{\mathbf{x}_j^k \in \left|N_{c,i}^*\right|} \frac{w_{ij}}{\sum_{\mathbf{x}_j^k \in \left|N_{c,i}^*\right|} w_{ij}} L_m(\mathbf{x}_i^c, \mathbf{x}_j^k; f). \tag{2.24}$$

The final ranked list loss for one anchor and all samples in the batch are as:

$$\begin{aligned} L_{RLL}(\mathbf{x}_i^c; f) &= L_p(\mathbf{x}_i^c; f) + L_N(\mathbf{x}_i^c; f), \\ L_{RLL}(\mathbf{X}; f) &= \frac{1}{N} \sum_{\forall c, \forall i} L_{RLL}(\mathbf{x}_i^c; f). \end{aligned} \tag{2.25}$$

2.5.2 Learning Rate

The learning rate is a hyper-parameter that controls how much the weights of the network vary to reduce the loss. The smaller the learning rate is, the slower the gradient descends. Even though a low learning rate ensures that the model not to miss any local minima, it implies that the model would take more time to converge, especially when the loss is stuck in a plateau region. Formula shown as below illustrates the principle of the gradient

descent algorithm:

$$new\_weight = previous\_weight + learning\_rate * gradient. \tag{2.26}$$

Gradient descent is a method to minimize the loss by iteratively updating weights with gradient. Figure 2.36 shows an illustration of the gradient descent algorithm.

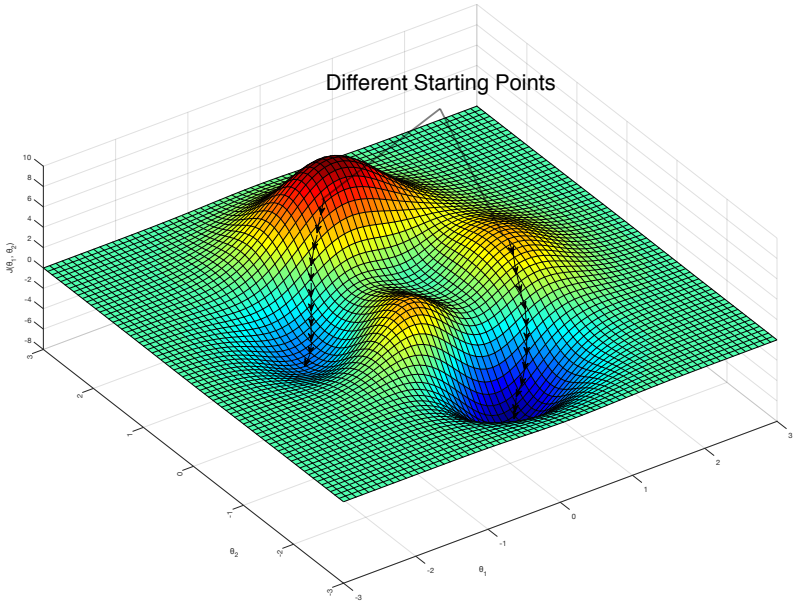


Figure 2.36. *Illustration of the gradient descent algorithm. z-axis denotes the loss $J(\theta_0, \theta_1)$, and x-axis and y-axis are for optimized parameters θ_0 and θ_1. The loss goes down along the negative direction of the gradient of the loss function and achieves the minima that can be global or local with different starting points.*

In this case, Θ represents the vector of parameters $(\theta_0, \theta_1)^T$ mathematically, and the gradients of the loss function $\Delta J(\Theta)$ with respect to weights are calculated as:

$$\Delta J(\Theta) = (\frac{\partial J(\Theta)}{\partial \theta_0}, \frac{\partial J(\Theta)}{\partial \theta_1}). \tag{2.27}$$

Weights are updated simultaneously with iterations until the loss converges, the formula of the gradient decent algorithm for this case is as:

$$\begin{aligned} \theta_0 &= \theta_0 - \alpha \frac{\partial J(\Theta)}{\partial \theta_0}, \\ \theta_1 &= \theta_1 - \alpha \frac{\partial J(\Theta)}{\partial \theta_1}. \end{aligned} \tag{2.28}$$

An excessive learning rate may directly make the model back and forth across the minima and never achieve it. While a too small learning rate makes the training slow and the model trapped in the local minima easily. The optimum learning rate is usually set through experiments and experience. Several learning rate methods that accelerate the model

convergence are presented in the following. Figure 2.37 shows an illustration of the performance of the loss with different learning rates.

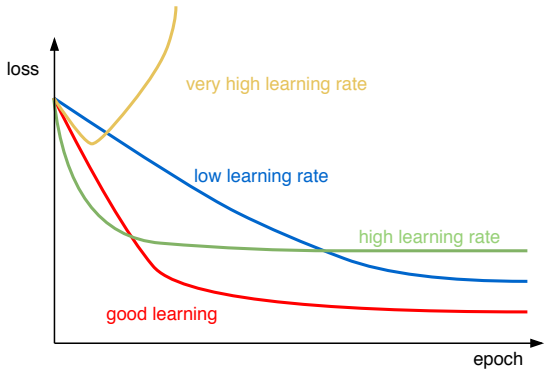


Figure 2.37. *Illustration of the performance of the loss with four different learning rates. The good learning rate (red curve) makes loss decrease fast at the beginning and converges to a stable value gently. The high learning rate (green curve) makes the model hover around the minima. The very high learning rate (yellow curve) makes the model not converge. The low learning rate (blue curve) makes the model converge slowly.*

Learning Rate Warm Up

The learning rate warm up method [12] is proposed for initializing the learning rate gradually: a relatively small learning rate is set with several epochs at the beginning of training until the model converges with a stable loss, and the training continues with different learning rate. The experiment is implemented with ResNet110 [12] on CIFAR-10 [18] dataset: the learning rate for warming up is set as 0.001 until the training error is below 80% with 400 epochs, and the training continues with the learning rate at 0.1. Finally, The result is satisfying with the method.

Facebook proposes another learning rate warm up method that the learning rate increases from small to large linearly within a few epochs [10]. It resolves the issue that a sudden increase of training error may occur when a small learning rate abruptly changes to a large one. Figure 2.38 shows an example of warming up the learning rate linearly:

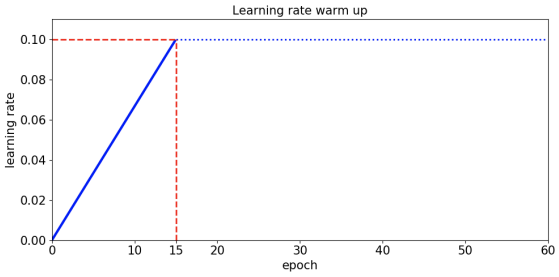


Figure 2.38. *Example of warming up linearly. The learning rate increases from 0 to 0.1 with 15 epochs according to the function* $\eta = \frac{1}{150}t$*, where* η *and* t *represent the learning rate and number of epochs, respectively.*

Learning Rate Decay

Learning rate decay is the approach that continues the training with a specific variation of the learning rate after warming up. It fastens the convergence of the model into a minima. There are three basic kinds of learning rate decay: piecewise decay, exponential decay and cosine decay shown in Figure 2.39.

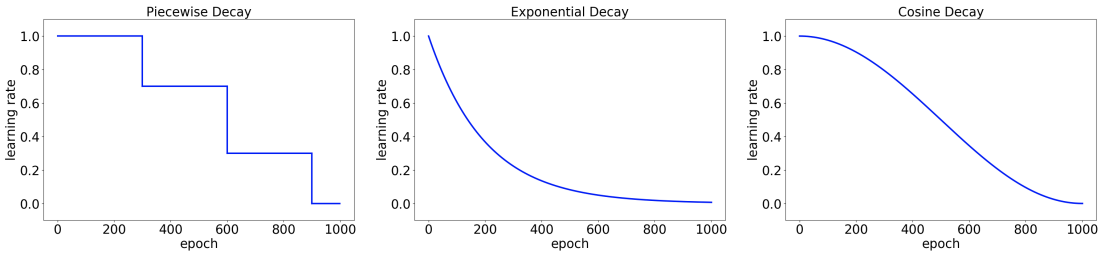


Figure 2.39. *Example of three different kinds of learning rate decay with 1,000 epochs. (a) Piecewise decay. The learning rate η decreases with the sequence of $\{1.0, 0.7, 0.3, 0\}$ at the epoch point of $\{0, 300, 600, 900\}$, respectively. (b) Exponential decay. The learning rate η decreases with the function $\eta = \exp(-0.005 \times t)$, where t is the number of epoch.(c) Cosine decay. The learning rate η decreases with the function $\eta = \frac{1}{2}(\cos(0.001\pi t) + 1)$.*

- The piecewise decay refers to the learning rate in different periods of epochs are with different constant. Figure 2.39a shows an example of the piecewise decay.
- The exponential decay refers to the decay learning rate in an exponential curve. Figure 2.39b shows an example of the exponential decay.
- The cosine decay refers to decay learning rate with a half period of cosine curve. Figure 2.39c shows an example of the cosine decay.

2.5.3 Optimization Algorithm

The original gradient descent algorithm requires the complete training set for calculating loss and updating weights with iterations. However, it is impractical for those tasks with a massive amount of dataset due to the limited hardware resource. Several optimization algorithms are proposed that trains the model with small batches of samples to converge effectively.

Stochastic Gradient Descent

Stochastic Gradient Descent (SGD) is proposed for resolving the issues mentioned previously: it calculates gradients and updates weights with a stochastic sample from the training set, which enables training samples to be appended during training. It is defined

as below, where Θ and $J(\Theta, x^{(i)}, y^{(i)})$ refer to the vector of weights and corresponding gradients with one training sample $(x^{(i)}, y^{(i)})$ in the stochastic batch:

$$\Theta = \Theta - \alpha \Delta J(\Theta, x^{(i)}, y^{(i)}). \tag{2.29}$$

Stochastic Gradient Descent has advantages compared to the original gradient descent that is named as Batch Gradient Descent:

- The convergence is fast since the Batch Gradient Descent contains redundant calculation, such as the gradient calculation of similar samples.
- Additional training samples can be added during training.
- It is more likely to jump out from a bad local minima and achieve a good one instead.

Mini-batch Gradient Descent

Mini-batch Gradient Descent is a compromise between the Stochastic Gradient Descent and the Batch Gradient Descent: it takes a subset from the training set for calculating gradients. It is defined as below, where $(x^{(i:i+n)}, y^{(i:i+n)})$ refers to the n samples in the mini-batch:

$$\Theta = \Theta - \alpha \Delta J(\Theta, x^{(i:i+n)}, y^{(i:i+n)}). \tag{2.30}$$

Mini-batch Gradient Descent only calculates the gradients of samples in a mini-batch in each epoch, which has not only effective calculation but also converges fast and stably. It is one of the mainstream algorithms at present.

Momentum

The Momentum algorithm is proposed for tackling the problem of previous algorithms that the updated direction of weights entirely depends on the direction of gradients, and that makes the training unstable. The concept of momentum in physics inspires the Momentum algorithm. It simulates the inertia of an object when it moves: the previous direction is retained in the current iteration, which enlarges the updated strength in the current iteration if the previous direction is consistent with the current one, and vice versa. So the Momentum algorithm integrates with historical gradients. The Gradient Descent with Momentum is defined as below, where v_t is the current updated item, and β is the

hyper-parameter for retention:

$$\begin{aligned} v_t &= \beta v_{t-1} + (1-\beta)\Delta J(\Theta), \\ \Theta &= \Theta - \alpha v_t. \end{aligned} \tag{2.31}$$

Adaptive Moment Estimation

Adaptive Moment Estimation (Adam) [17] is a self-adaptive algorithm. The actual learning rate is dynamically adjusted through the first and second moment estimates of the gradient. It makes the actual learning rate within a certain range with bias correction and the training stable. The first moment estimate m_t and second moment estimate v_t are defined as below, where g_t and g_t^2 refer to the gradient and its squire with dot production:

$$\begin{aligned} m_t &= \beta_1 m_{t-1} + (1-\beta_1) g_t, \\ v_t &= \beta_2 v_{t-1} + (1-\beta_2) g_t^2. \end{aligned} \tag{2.32}$$

The first moment estimate m_t and second moment estimate v_t are initialized with a bias to 0, so a bias correction is applied:

$$\begin{aligned} \hat{m_t} &= \frac{m_t}{1-\beta_1^t}, \\ \hat{v_t} &= \frac{v_t}{1-\beta_2^t}. \end{aligned} \tag{2.33}$$

Finally, the Adaptive Moment Estimation is defined as, where the hyper-parameters β_1, β_2 and ϵ are set as 0.9, .999 and 10e-8 respectively according to the author's suggestion:

$$\Theta = \Theta - \frac{\alpha}{\sqrt{\hat{v_t}} + \epsilon} \hat{m_t}. \tag{2.34}$$

2.5.4 Forward-propagation and Back-propagation

Forward-propagation and back-propagation are the algorithms that cooperate to optimize weights by minimizing the loss in an efficient way.

- Forward-propagation is the process flow that calculates the result, which is from the input layer with linear operations and activation functions. Neurons among layers are connected with weight vector $\boldsymbol{\theta}_i^{(l)}$, in which, l refers to the notation of layer and i is the index of a neuron in the $l+1$ layer. The forward-propagation process is

summarized as:

1. i-th neuron in the j-th layer $z_i^{(j)}$ receives the output $a_i^{(j-1)}$ from previous layer:

$$z_i^{(j)} = \boldsymbol{\theta}_i^{(j-1)} a_i^{(j-1)}. \tag{2.35}$$

2. After passing through the activation function $y = g(x)$, the output of the i-th neuron in j-th layer $a_i^{(j)}$ is obtained as:

$$a_i^{(j)} = g(z_i^{(j)}). \tag{2.36}$$

The prediction $h_{\boldsymbol{\Theta}}(x)$ in Figure 2.40 could be obtained, which shows an example of forward-propagation with a fully-connected neural network:

$$\begin{aligned} a_i^{(1)} &= x_i, i \in \{1, \ldots, 400\}, \\ z_i^{(2)} &= \boldsymbol{\theta}_i^{(1)} \boldsymbol{a}^{(1)}, i \in \{1, \ldots, 25\}, \\ a_i^{(2)} &= g(z_i^{(2)}), i \in \{1, \ldots, 25\}, \\ z_1^{(3)} &= \boldsymbol{\theta}_1^{(2)} \boldsymbol{a}^{(2)}, \\ a_1^{(3)} &= g(z_1^{(3)}), \\ h_{\boldsymbol{\Theta}}(x) &= a_1^{(3)}. \end{aligned} \tag{2.37}$$

In the training process, the loss $L(\boldsymbol{\Theta})$ can be calculated with the prediction $h_{\boldsymbol{\Theta}}(x)$ and the ground truth $\boldsymbol{Y}$.

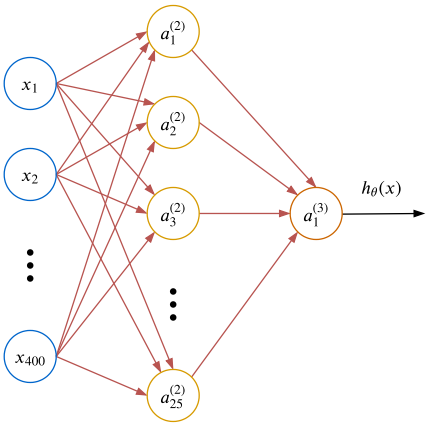


Figure 2.40. *Example of forward-propagation. The prediction $h_{\boldsymbol{\Theta}}(x)$ is obtained with the forward-propagation algorithm.*

- The back-propagation algorithm is to calculate the gradients of weights in an efficient approach that the model can be optimized by the gradient descent algorithm iteratively. Figure 2.41 shows an example of back-propagation based on Figure 2.40.

  The goal of back-propagation is to calculate gradients for the weights $\boldsymbol{\Theta}$ with follow-

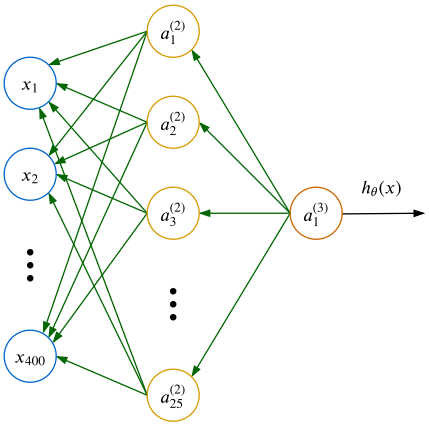


Figure 2.41. *Example of back-propagation. The gradients matrices for the first and second layers $\Theta^{(1)}$ and $\Theta^{(2)}$ can be obtained with the back-propagation algorithm after the forward-propagation.*

ing the Chain Rule of calculus. The gradients matrices for the weights in the first and second layers $\Theta^{(1)}$ and $\Theta^{(2)}$ in Figure 2.41 are calculated as:

$$\begin{aligned}\frac{\partial L(\boldsymbol{\Theta})}{\partial \Theta^{(2)}} &= \frac{\partial L(\boldsymbol{\Theta})}{\partial a_1^{(3)}} \frac{\partial a_1^{(3)}}{\partial g^{(3)}} \frac{\partial g^{(3)}}{\partial z_1^{(3)}} \frac{\partial z_1^{(3)}}{\partial \Theta^{(2)}}, \\ \frac{\partial L(\boldsymbol{\Theta})}{\partial \Theta^{(1)}} &= \frac{\partial L(\boldsymbol{\Theta})}{\partial a_1^{(3)}} \frac{\partial a_1^{(3)}}{\partial g^{(3)}} \frac{\partial g^{(3)}}{\partial z_1^{(3)}} \frac{\partial z_1^{(3)}}{\partial \boldsymbol{a}^{(2)}} \frac{\partial \boldsymbol{a}^{(2)}}{\partial \boldsymbol{g}^{(2)}} \frac{\partial \boldsymbol{g}^{(2)}}{\partial \boldsymbol{z}^{(2)}} \frac{\partial \boldsymbol{z}^{(2)}}{\partial \Theta^{(1)}}.\end{aligned} \tag{2.38}$$

Furthermore, each term in Equation 2.38 has been calculated through the forward-propagation in one iteration due to the derivatives calculated with the linear operation and pre-defined activation functions. Then, $\boldsymbol{\Theta}$ could be updated by the gradient descent algorithm, in which $\Theta^{(1)}$ and $\Theta^{(2)}$ should be updated simultaneously:

$$\begin{aligned}\Theta^{(2)} &= \Theta^{(2)} + \alpha \frac{\partial L(\boldsymbol{\Theta})}{\partial \Theta^{(2)}}, \\ \Theta^{(1)} &= \Theta^{(1)} + \alpha \frac{\partial L(\boldsymbol{\Theta})}{\partial \Theta^{(1)}}.\end{aligned} \tag{2.39}$$

Forward-propagation and back-propagation are interdependent and operational alternatively in the training of a neural network to optimize the weights until the convergence of the loss.

2.6 Evaluation Metric

This section introduces several metrics for evaluating the quantitative performance of the model in different aspects, which make the evaluation comprehensive.

2.6.1 Distance

The distance metric is the most crucial methodology to evaluate the similarity between two vectors. Euclidean distance and cosine similarity related to our project are introduced in the following.

Euclidean Distance

Given two n dimension vectors $\boldsymbol{x} = (x_1, \ldots, x_n)$ and $\boldsymbol{y} = (y_1, \ldots, y_n)$, the euclidean distance between them is:

$$euclidean\_distance(\boldsymbol{x}, \boldsymbol{y}) = \sqrt{\sum_{i=1}^{n}(x_i - y_i)^2}. \tag{2.40}$$

In matrix representation, it is as:

$$euclidean\_distance(\boldsymbol{x}, \boldsymbol{y}) = \sqrt{(\boldsymbol{x} - \boldsymbol{y})^T(\boldsymbol{x} - \boldsymbol{y})}. \tag{2.41}$$

Cosine Similarity

Given two n dimension vectors $\boldsymbol{x}(x_1, ..., x_n)$ and $\boldsymbol{y}(y_1, ..., y_n)$, the cosine similarity between them is:

$$cosine\_similarity(\boldsymbol{x}, \boldsymbol{y}) = \cos\theta = \frac{\sum_{i=1}^{n}(x_i \times y_i)}{\sqrt{\sum_{i=1}^{n} x_i^2} \times \sqrt{\sum_{i=1}^{n} y_i^2}}. \tag{2.42}$$

In matrix representation, it is as:

$$cosine\_similarity(\boldsymbol{x}, \boldsymbol{y}) = \cos\theta = \frac{\boldsymbol{x} \cdot \boldsymbol{y}}{\|\boldsymbol{x}\| \cdot \|\boldsymbol{y}\|}. \tag{2.43}$$

Euclidean distance reflects the absolute difference of numerical characteristics, while cosine similarity is sensitive to the direction difference between two vectors. Notably, all dimensions on the euclidean distance require to be at the same scale to make the result reliable.

2.6.2 Mean Average Precision

Mean Average Precision (mAP) calculates the mean of average precisions from multiple query samples, in which the average precision refers to the average value of precisions from positive predictions belong to one query sample. The formula of the mAP is as below, where N denotes the number of query samples:

$$mAP = \frac{1}{N}\sum_{i=1}^{N} AP_i. \tag{2.44}$$

Precision and Recall

Precision is the ratio between correctly predicted samples and all predicted samples:

$$Precision = \frac{True\ Positive}{True\ Positive + False\ Positive}.$$

Recall is the ratio between correctly predicted samples and all correct samples:

$$Recall = \frac{True\ Positive}{True\ Positive + False\ Negative}.$$

Figure 2.42 shows an example of the mAP calculation with two queries: query 1 and query 2, where the corresponding predictions contain 5 and 3 positives, respectively:

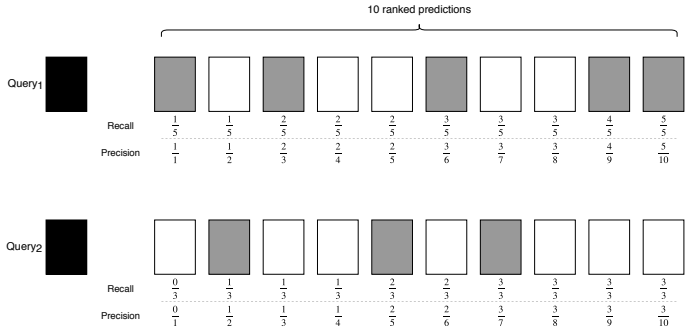


Figure 2.42. *Example of the mAP calculation with two queries. Black squares represents the query 1 and query 2. Ten ranked predictions for each query, where white and gray squares stand for the incorrect and correct predictions, respectively.*

the Average Precision of query 1 and query 2 in Figure 2.42 are calculated as:

$$Average\ Precision\ query\ 1 = (\frac{1}{1} + \frac{2}{3} + \frac{3}{6} + \frac{4}{9} + \frac{5}{10})/5 = 0.62,$$
$$Average\ Precision\ query\ 2 = (\frac{1}{2} + \frac{2}{5} + \frac{3}{7})/3 = 0.44.$$

Finally, the mean Average Precision of these two queries is:

$$mean\ Average\ Precision = (0.62 + 0.44)/2 = 0.53.$$

Besides, Average Precision is the area under Precision-Recall curve, Figure 2.43 illustrates the area under Precision-Recall based on the example in Figure 2.42.

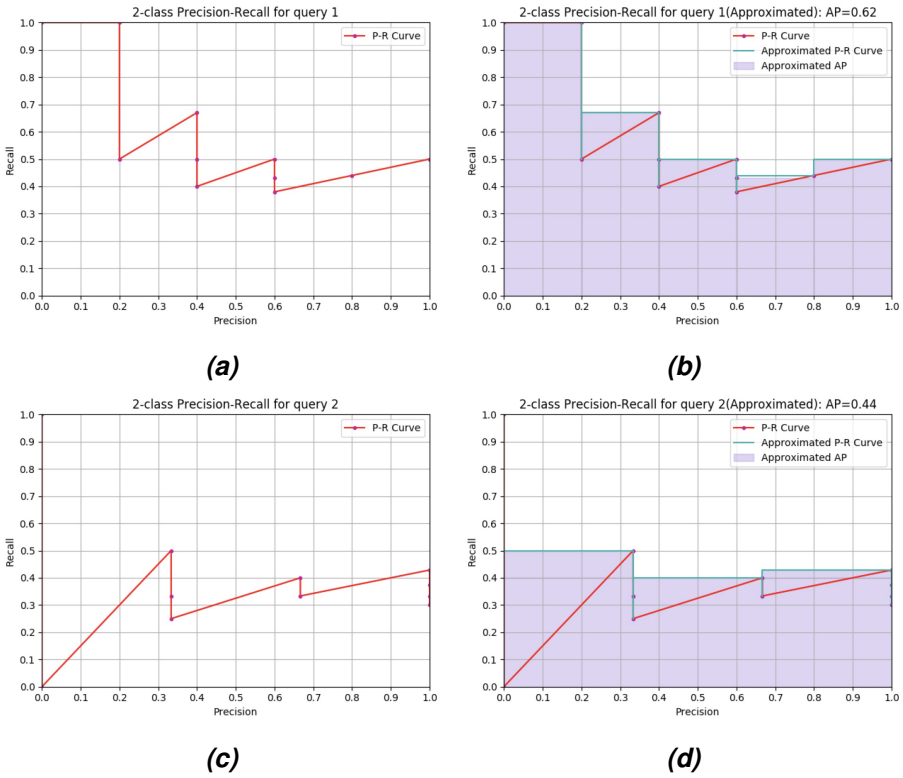


Figure 2.43. *Illustration of the area under Precision-Recall curve based on the example in Figure 2.42. (a) and (c) are the original Precision-Recall curves of query 1 and 2, which are drawn with precisions in the x-axis and recalls in the y-axis. (b) and (d) are the approximated areas (purple area) of them that it approximates the recall as constant within each interval of precision, where the intervals are* $\frac{1}{5}$ *and* $\frac{1}{3}$*.*

2.6.3 Cumulative Matching Characteristics

Cumulative Matching Characteristics (CMC) is a method of top-k shot probability to evaluate accuracy, in which top-k refers to the first k samples with ranking rules. The model makes the predictions according to the query sample and ranks them with the similarities to the query. The CMC top-k accuracy for the query is defined as:

$$Accuracy = \begin{cases} 1, \text{if the top-k ranked predictions contain query identity,} \\ 0, \text{otherwise.} \end{cases} \tag{2.45}$$

2.7 Cross Validation

The validation set is for evaluating whether a model is over-fitting with a satisfying performance in training. The main idea of cross validation is to separate the validation set from the dataset for the over-fitting verification and utilize the rest part for training. This section introduces three methods of cross validation: the Holdout method, K-fold Cross Validation, and Leave-One-Out Cross Validation.

2.7.1 Hold-Out Method

The Hold-Out method randomly split the training set for training and validation with a specific ratio. The Hold-Out is the simplest validation method that is effective if the dataset is massive. The proportion of validation set in the original training set is usually at 20%-30%. Strictly, The Hold-Out method is not a method of cross validation since there is no "cross" idea in it. Besides, the validation result is highly related to the random split of the validation set, which may have a significant difference in performance with different separation. Figure 2.44 is an illustration of the Hold-Out method.

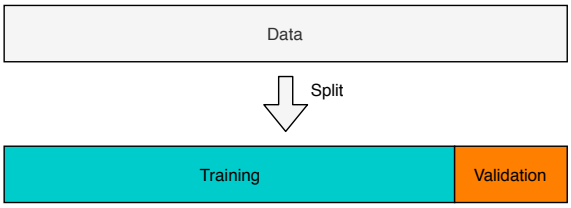


Figure 2.44. *Illustration of the Hold-Out. The original training set is split into the training part (blue) and the validation part (orange).*

2.7.2 K-fold Cross Validation

The K-fold Cross Validation is to split the original training set into K folds of subsets evenly, where one of them is for validation, and rest K-1 subsets are for training. Besides, each subset would be for validation sequentially, and the final result is calculated with the mean of K validation results. The K-fold Cross Validation is beneficial since the whole dataset is used for both training and validation and is appropriate for the small dataset. K is usually greater than two. Figure 2.45 is an illustration of K-fold Cross Validation.

2.7.3 Leave-One-Out Cross Validation

The Leave-One-Out Cross Validation (LOO-CV) is a specific case of the K-fold Cross Validation. Given N samples in a dataset and the K is set as N, that is, it splits the dataset

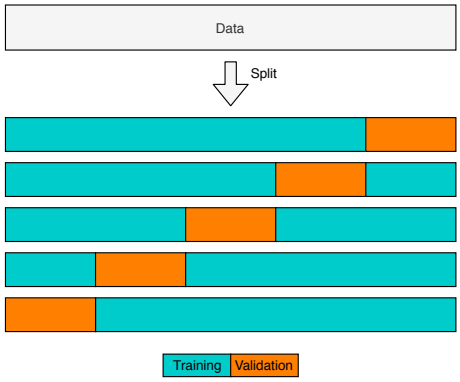


Figure 2.45. *Example of K-fold Cross Validation. The original training set is split evenly into five folds of subsets, where the blue and orange squares represent for training and validation subsets.*

into N subsets, which takes one sample for the validation and the rest N-1 samples for the training. The Leave-One-Out Cross Validation has several advantages:

- Nearly all samples in the dataset are utilized for training that the distribution of the training part is close to the original training set, which makes the result reliable.
- The splitting of the training and validation parts is without random factors, which makes the result reproducible.

However, the shortcoming is that the computational complexity is high since the training requires to be implemented as many times as the number of samples with nearly full dataset wherefore it is inappropriate for the large dataset. Figure 2.46 is an illustration of the Leave-One-Out Cross Validation.

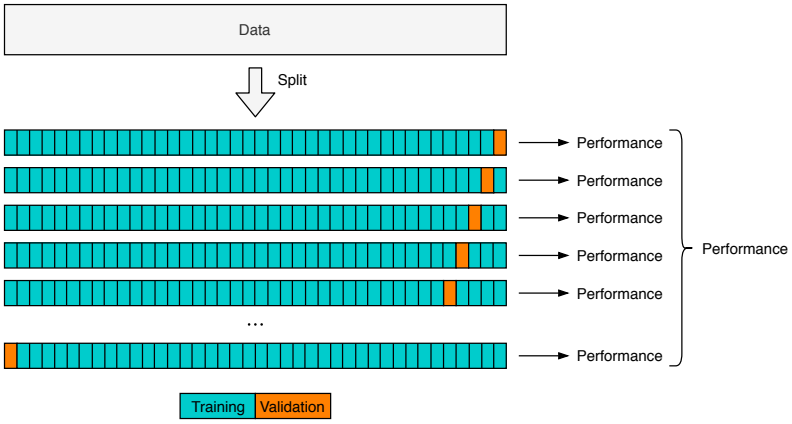


Figure 2.46. *Illustration of the Leave-One-Out Cross Validation. A dataset contains N samples is split into N folds evenly, where the blue and orange squares represent for training and validation subsets.*

3 RESEARCH METHODOLOGY

This chapter introduces the actual implementation of person re-identification task, which includes essential components, training and inference process for the baseline model BNNeck[25] and our proposed method SplitReID[1].

3.1 Dataset

In the experiments, three person re-identification datasets are utilized for training and evaluation: Market-1501 [44], DukeMTMC-reID [32] and MSMT17 [41]. Their introduction is presented below and the basic statistics are summarized in Table 3.1.

3.1.1 Market-1501

Market-1501 [44] person dataset is collected on the campus of Tsinghua University with 1,501 pedestrians and 32,668 bounding boxes across six cameras. Each pedestrian is photographed by at least two different cameras. The training set contains 751 pedestrians with 12,936 images, and the test set contains 750 pedestrians and the distractors annotated with ID “0000” with 15,913 images, respectively. The query set contains 3,368 images that are labeled manually, and the test set detected by DPM is as the gallery.

3.1.2 DukeMTMC-reID

DukeMTMC-reID [32] is a subset of DukeMTMC person dataset, which is collected across eight cameras. It contains 1,404 people who appear in more than two cameras, and the rest 408 pedestrians are the distractors that appear only in one camera. The training set contains 702 pedestrians with 17,661 images, and the test set contains 1,110 pedestrians with 16,522 images. 2,228 images are randomly selected from the test set for the query, and the test set is as the gallery

[1] The paper “SplitReID: Target Re-Identification Made Easy” has been submitted to WACV 2020.

3.1.3 MSMT17

MSMT17 [41] is a massive person dataset compared to Market-1501 and DukeMTMC-reID. It is collected across twelve outdoor and three indoor cameras with different climate conditions. MSMT17 contains 126,441 images from 4,101 pedestrians. The training and test set is randomly split into the ratio at 3:1. The training set contains 1,041 pedestrians with 32,621 images and the test set contains 93,820 images from 3,060 pedestrians. Furthermore, 11,659 and 82,161 images are selected for the query and gallery, respectively.

Dataset	Market-1501	DukeMTMC-reID	MSMT17
Train and valid samples	12,936	16,522	32,621
Train and valid identities	751	702	1,041
Test query samples	3,368	2,228	11,659
Test gallery samples	15,913	17,661	82,161
Test identities	751	1,110	3,060
Cameras	6	8	15

Table 3.1. *Comparison of person and vehicle re-identification datasets.*

3.2 Baseline

Our proposed method is extended from the approach introduced by Luo *et al*. [25], which is considered as the baseline for our own approach. The essential components and details for training process of the baseline model is introduced below.

3.2.1 Backbone

In order to be consistent with others' works for comparison, ResNet50 [12] is selected as the backbone model for the baseline model, which is initialized with pre-trained weights on ImageNet [5]. Furthermore, the original ResNet50 model is modified slightly for the backbone with: firstly, the last fully-connected layer for the classification is removed since the rest part is utilized for the metric learning with feature vectors. Secondly, the parameter "stride" of the first convolutional layer in the last building block is set to 1 from 2. So, the output feature maps are double-sized from the original set, and it is beneficial for performance improvement with the higher spatial resolution.

3.2.2 BNNeck structure

The BNNeck structure proposed in [25] is shown in Figure 3.1. The feature vector f_t produced from the global average pooling (GAP) layer is utilized for the training with triplet loss [13]. Besides, a batch normalization layer is added after the GAP layer to yield feature vector f_i for the inference with cosine similarity. Lastly, the fully-connected layer after the batch normalization layer is set with the number of the identities in the training set. The output vector f_{ce} is for the training with cross-entropy loss.

Notably, the fully-connected layer is without bias term but includes a kernel, which is initialized from a normal distribution with mean 0.0 and standard deviation 0.001. The final loss function consists of two losses, where the coefficients of them equal 1.

Luo *et al*. [25] adopt the batch normalization layer to separate two losses into two feature embedding spaces, which help improve the performance.

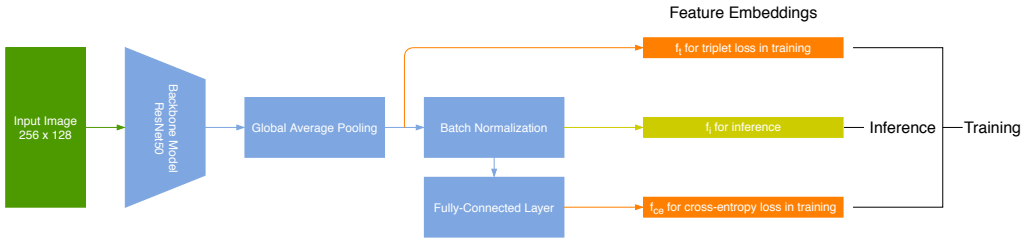


Figure 3.1. *The BNNeck structure proposed in [25]. In the training procedure, the model is optimized with two losses: triplet loss and categorical cross-entropy loss with different feature embeddings. In the inference procedure, the output tensor generated from the batch normalization layer is as the feature embedding.*

3.2.3 Mini-batch

Each mini-batch consists of samples that are from both the same and different identities to give full play to the advantages of the triplet loss function [13, 33] so that positives and negatives could be exploited with triplet loss.

P denotes the number of identities and K represents the number of samples with each identity so that the batch size is $B = P \times K$. All samples with the same identity would be abandoned in the training procedure progresses if the number of remaining samples from that identity is smaller than K. Furthermore, the training process of one epoch would complete if the number of remaining identities in the mini-batch is lower than P.

3.2.4 Data pre-processing and augmentation

Firstly, the data pre-processing is to resize the input images into target resolution as 256×128 (height $\times$ width). Secondly, the resized image is processed with the pre-processing

method provided by the pre-trained backbone model in frameworks.

Furthermore, several basic transformations for augmentation are applied for the resized images: images are horizontally flipped with 50% probability. Crop the images that are zero-padding with margins into the target resolution at random. Last but not the least, random erasing [49] erases an area of images with ImageNet [5] mean pixel value in the training process to improve the robustness of the model against the occlusions in practical cases. The Original image and random-erased images from Market-1501 [44] dataset are shown in Figure 3.2.

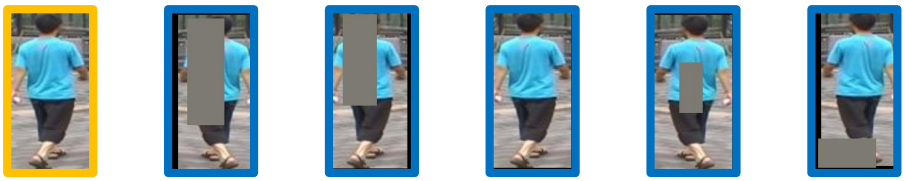


Figure 3.2. *Illustration of data augmentation that are processed sequentially with horizontal flipping, zero padding, random cropping and random erasing. Images with orange border and with blue border are original sample and augmented samples, respectively.*

3.2.5 Training aspects

Learning Rate

Luo *et al*. [25] adopt a variable learning rate. It utilizes the learning rate warm up method within several epochs in the first phase, which is from 0 to a relatively large learning rate. Then, the learning rate is divided by ten gradually in the following three phases. Figure 3.3 shows the learning rate scheduler.

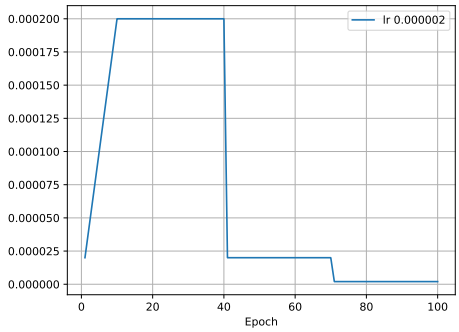


Figure 3.3. *The learning rate scheduler. The warmup strategy prevents the gradient distribution from being distorted to avoid the convergence problem [22]. Besides, sequentially decreasing the learning promotes the model converge into a better local minimum.*

Label-smoothing

The label-smoothing [38] method is adopted for the categorical cross-entropy loss to regularize the one-hot encoding label. $y \in \{1, 2, \ldots, N\}$ is the ground truth label of a sample, and the corresponding one-hot coding label $\boldsymbol{q}(i)$ equals 1 if the index i is compatible with label y, otherwise it equals 0. The label-smoothing method utilizes an hyper-parameter $\epsilon \in (0, 1)$ to balance the one-hot encoding label, which is calculated as:

$$\boldsymbol{q}'(i) = (1 - \epsilon)\boldsymbol{q}(i) + \frac{\epsilon}{N}. \tag{3.1}$$

The reason for performance improvement is understandable; *i.e.*, given a binary classification case, the prediction would never achieve 0 or 1 so that the number of weight w will continuously increase and the over-fitting may occur. While the label-smoothing is applied for the label, the training procedure would complete once the prediction achieves the "smoothed" ground truth. Label-smoothing method makes the model less confident with its predictions and alleviates the over-fitting issue. The hyper-parameter ϵ is set to 0.1. Figure 3.4 shows an example of the label-smoothing example.

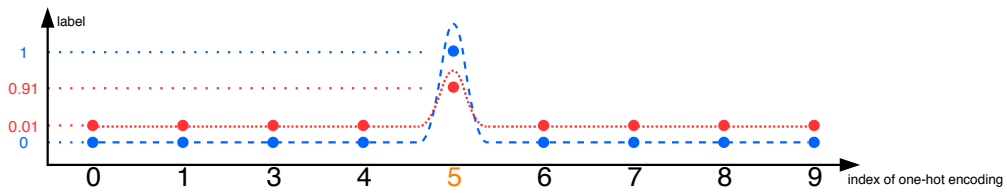


Figure 3.4. *Example of label-smoothing with ten classes. Blue dots indicate the original one-hot encoding label that the ground truth is the fifth class shown with orange. Red dots represent the smoothed label.*

L_2 regularization

ℓ_2 regularization is adopted for the trainable parameters, which are the weight/bias terms in the convolutional and fully-connected layers, and the γ/β terms in the batch normalization layers. The expression is as:

$$L_\lambda(\boldsymbol{w}) = L(\boldsymbol{w}) + \lambda \|\boldsymbol{w}\|_2^2, \tag{3.2}$$

in which $L(\boldsymbol{w})$ is the loss without regularization and λ is the regularization factor to control the "punishment". ℓ_2 regularization forces the model to train with smaller number of weights so that the over-fitting problem is suppressed. The regularization factor λ is set to 0.0005.

3.3 SplitReID Structure

This section introduces our proposed method SplitReID in the training and inference procedures, which is inspired by the BNNeck structure and aims to improve the performance by separating two losses into two different feature embedding spaces.

3.3.1 Training with SplitReID

The SplitReID structure shown in Figure 3.5 contains a splitting layer after the global average pooling layer to divide the output tensor into two parallel partitions compared to the BNNeck structure illustrated in Figure 3.1. Instead of the straight flow from input to output, two splitting feature vector slices are partitioned sequentially and evenly from the original one, who owns the same dimensionality and could be the input for the following two parallel branches.

The lower branch shown in Figure 3.5 takes the categorical cross-entropy loss for the output. The batch normalization [16] layer yields the normalized feature vector f_{i_ce} from the slice. Then the fully-connected layer makes the classification according to the identities in the training set. The structure in this branch is proposed in [43], which can provide more a steady gradient. The upper branch in Figure 3.5 is processed with the triplet loss. Another slice is fed into the triplet loss directly without extra manipulation, that the branch is implemented with the suggestion in [13, 19].

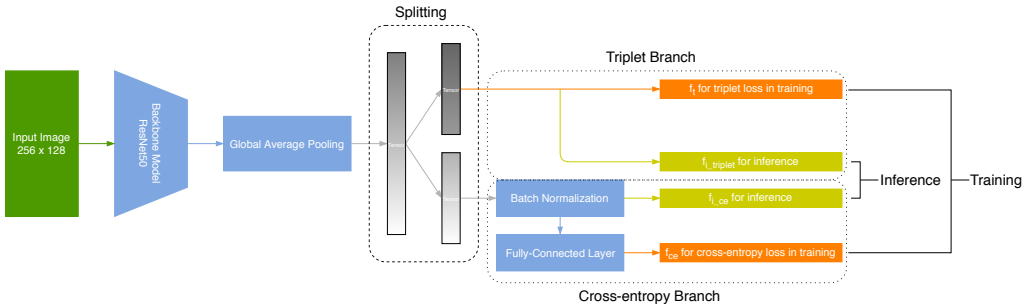


Figure 3.5. *The SplitReID structure splits the feature vector generated from the global average pooling layer evenly into two separate branches: one is for categorical cross-entropy loss and another is for triplet loss.*

3.3.2 Inference with SplitReID

When it comes to the inference procedure of the SplitReID. Given an input image $\mathbf{X}$, we denote $\mathbf{y}_{i\_ce} = f_{i\_ce}(\mathbf{X})$ and $\mathbf{y}_{i\_triplet} = f_{i\_triplet}(\mathbf{X})$ as the feature vectors extracted from the categorical cross-entropy loss branch and the triplet loss branch, respectively.

Moreover, the query and gallery image sets are denoted as $\mathcal{Q} = \{\mathbf{Q}_1, \mathbf{Q}_2, \ldots, \mathbf{Q}_m\}$ and

$\mathcal{G} = \{\mathbf{G}_1, \mathbf{G}_2, \ldots, \mathbf{G}_n\}$, so the corresponding feature vectors with respect to two branches are:

$$\begin{aligned} F_{i\_ce}(\mathcal{Q}) &= \{f_{i\_ce}(\mathbf{Q}_1), \ldots, f_{i\_ce}(\mathbf{Q}_m)\}, \\ F_{i\_triplet}(\mathcal{Q}) &= \{f_{i\_triplet}(\mathbf{Q}_1), \ldots, f_{i\_triplet}(\mathbf{Q}_m)\}, \\ F_{i\_ce}(\mathcal{G}) &= \{f_{i\_ce}(\mathbf{G}_1), \ldots, f_{i\_ce}(\mathbf{G}_n)\}, \\ F_{i\_triplet}(\mathcal{G}) &= \{f_{i\_triplet}(\mathbf{G}_1), \ldots, f_{i\_triplet}(\mathbf{G}_n)\}. \end{aligned} \tag{3.3}$$

When considering two images $\mathbf{X}_1$ and $\mathbf{X}_2$, we have denotations: $d_{i\_ce}(f_{i\_ce}(\mathbf{X}_1), f_{i\_ce}(\mathbf{X}_2))$ and $d_{i\_triplet}(f_{i\_triplet}(\mathbf{X}_1), f_{i\_triplet}(\mathbf{X}_2))$ that refer to the distance similarities between the corresponding feature vectors. Besides, the cosine similarity is adopted according to practical cases in some works.

Consequently, the m-by-n distance matrix $\mathbf{D}_{i\_ce}$ between query and gallery sets could be calculated with d_{i_ce} from the outer product of $F_{i\_ce}(\mathcal{Q})$ and $F_{i\_ce}(\mathcal{G})$. The other m-by-n distance matrix $\mathbf{D}_{i\_triplet}$ for triplet loss can be obtained in the same way.

To integrate two distance matrices, firstly, we adopt a standardization for the distance matrices since they are calculated from two feature embedding spaces that may have a significant difference of scale in value, which is centralized to the origin and scaled to the interquartile range. Secondly, we take the arithmetic mean matrix of two standardized distance matrices as the final inference result of the SplitReID model.

4 RESULTS AND ANALYSIS

This chapter introduces the experimental result and corresponding analysis of our proposed SplitReID and other state-of-the-art methods on three person datasets. The demonstration with our SplitReID structure on three datasets is shown at the end of the chapter.

4.1 Results of person re-identification

4.1.1 Results on Market-1501 and DukeMTMC-reID

Table 4.1 shows the comparison among the BNNeck structure, our proposed SplitReID structure and recent works on the Market-1501 [44] and DukeMTMC-reID [32] datasets. ABD-Net [2] achieves the best performance on both datasets, which is composed of a relatively complicated network with attention mechanism and higher resolution images. The BNNeck structure proposed by Luo *et al*. [25] achieves 86.6% and 77.2% mAP on Market-1501 and DukeMTMC-reID datasets, respectively. Furthermore, our SplitReID method promotes a noteworthy improvement at 87.2% and 77.9% mAP.

Method Name	Backbone	Image Size	Market-1501		DukeMTMC-reID	
			mAP	rank-1	mAP	rank-1
CNN Embedding [47]	ResNet50	227 x 227	59.9	79.5	-	-
GAN [46]	ResNet50	224 x 224	66.1	84.0	47.1	67.7
PAN [48]	ResNet50	224 x 224	69.3	86.7	51.5	71.6
CamStyle [50]	ResNet50	256 x 128	71.6	89.5	57.6	78.3
PCB+RP [36]	ResNet50	384 x 128	81.6	93.8	69.2	83.3
BoT Baseline [25]	ResNet50	256 x 128	85.9	94.5	76.4	86.4
DG-Net [45]	ResNet50	256 x 128	86.0	94.8	74.8	86.6
ABD-Net [2]	ResNet50	384 x 128	**88.3**	**95.6**	**78.6**	**89.0**
BNNeck structure	ResNet50	256 x 128	86.6	94.1	77.2	86.9
SplitReID structure	ResNet50	256 x 128	87.2	94.3	77.9	87.6

Table 4.1. *Results on Market-1501 [44] and DukeMTMC-reID [32].*

4.1.2 Results on MSMT17

Table 4.2 illustrates the comparison of ours and some methods on the MSMT17 dataset, which is more challenging. Similarly, our proposed SplitReID achieves a competitive performance at 52.4% mAP and 74.6% rank-1 accuracy, that it is the highest performance among the methods based on the global feature.

Method Name	Backbone	Image Size	MSMT17	
			mAP	rank-1
GoogLeNet [37, 41]	GoogLeNet	-	23.0	47.6
PDC [35, 41]	-	-	29.7	58.0
GLAD [41, 42]	-	-	34.0	61.4
ABD-Net [2]	ResNet50	384 x 128	**60.8**	**82.3**
BNNeck structure	ResNet50	256 x 128	51.1	74.4
SplitReID structure	ResNet50	256 x 128	52.4	74.6

Table 4.2. *Results on MSMT17 [41].*

4.1.3 Results on ResNet and DenseNet

For further study about the performance with different backbone models, DenseNet[15] series models are selected for the comparison to ResNet50. Besides, the experiment implementation is based on the SplitReID structure with the input image size at 256×128 (height $\times$ width).

Table 4.3 compares the performance with different backbone models alongside with their number of parameters in the inference. As we can see, the DenseNet201 with a smaller number of parameters exceeds the performance of ResNet50 on three datasets, which can be one of the directions to improve the performance in the late work.

Backbone	Parameters	Market-1501		DukeMTMC-reID		MSMT17	
		mAP	rank-1	mAP	rank-1	mAP	rank-1
ResNet50	23.60M	87.2	94.3	77.9	87.6	52.4	74.6
DenseNet121	7.04M	85.5	94.1	76.5	87.8	52.0	76.4
DenseNet169	12.65M	86.1	94.3	77.1	88.2	52.4	75.9
DenseNet201	18.33M	**87.2**	**94.5**	**78.3**	**88.7**	**55.2**	**77.9**

Table 4.3. *Results on three datasets with ResNet50 [12] and DenseNets [15].*

4.2 Parameter Reduction

Table 4.4 shows the statistics of the number of parameters between the BNNeck structure and our SplitReID structure on three datasets. The SplitReID structure requires fewer parameters since it benefits from the splitting branches, that the dimensionality of the feature vector for the categorical cross-entropy loss halves into the fully-connected layer compared to the BNNeck structure.

Dataset	Total Parameters (in millions)		
	BNNeck	SplitReID	Reduction
Market-1501	25.1	24.3	3.2%
DukeMTMC-reID	25.0	24.3	2.8%
MSMT17	25.7	24.6	4.3%

Table 4.4. *Parameter reduction in the inference procedure.*

4.3 Demonstration

Figure 4.1 shows a demonstration of predictions with our SplitReID model on the three datasets.

Figure 4.1. *Examples of query and top-5 gallery samples on the three datasets predicted by our SplitReID model. Images with orange, green and red borders are query samples, correct and erroneous predictions, respectively.*

5 CONCLUSION

In this work, the pre-trained model ResNet50 provided by TensorFlow [1] is adopted to train and infer on the three person datasets: Market-1501, DukeMTMC-reID, and MSMT17. For the person re-identification task, a trained neural network model retrieves images that are from the same person across multiple cameras in the gallery set when given a query image. The performance is evaluated with mean Average Precision (mAP), and Cumulative Matching Characteristic (CMC) rank-k accuracy.

The BNNeck structure aims to train the model with multi-loss in two separated feature embedding spaces, which inspires us to propose the SplitReID structure. The SplitReID improves the independence of each feature embedding space that the model could be optimized by taking full advantage of different losses.

In the experiment, it has been demonstrated that our proposed SplitReID structure outperforms the BNNeck structure, and reaches a competitive performance on the three person re-identification datasets with a relatively simple network structure. Furthermore, in contrast to the BNNeck structure, our proposed SplitReID structure requires fewer parameters that reduce the computation complexity. Besides, with the replacement of Denset201, the SplitReID hits a new high on three datasets. Shortly, the performance of our SplitReID structure could be improved with more separated embedding spaces and the substitution of more powerful backbone models.

The demonstration in Chapter 4 shows impressive results on three datasets. Although some issues related to illumination variation and occlusion are to be tackled in the future, it still has tremendous potential in security fields, such as Intelligent Trace System in public.

REFERENCES

[1] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard et al. Tensorflow: A system for large-scale machine learning. *12th USENIX Symposium on Operating Systems Design and Implementation*. 2016, 265–283.

[2] T. Chen, S. Ding, J. Xie, Y. Yuan, W. Chen, Y. Yang, Z. Ren and Z. Wang. ABD-Net: Attentive but Diverse Person Re-Identification. *arXiv preprint arXiv:1908.01114* (2019).

[3] C. Cortes and V. Vapnik. Support-vector networks. *Machine learning* 20.3 (1995), 273–297.

[4] N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. 2005.

[5] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li and L. Fei-Fei. Imagenet: A large-scale hierarchical image database. *2009 IEEE conference on computer vision and pattern recognition*. Ieee. 2009, 248–255.

[6] J. Deng, J. Guo, N. Xue and S. Zafeiriou. Arcface: Additive angular margin loss for deep face recognition. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2019, 4690–4699.

[7] P. F. Felzenszwalb, R. B. Girshick, D. McAllester and D. Ramanan. Object detection with discriminatively trained part-based models. *IEEE transactions on pattern analysis and machine intelligence* 32.9 (2009), 1627–1645.

[8] R. Girshick. Fast r-cnn. *Proceedings of the IEEE international conference on computer vision*. 2015, 1440–1448.

[9] R. Girshick, J. Donahue, T. Darrell and J. Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2014, 580–587.

[10] P. Goyal, P. Dollár, R. Girshick, P. Noordhuis, L. Wesolowski, A. Kyrola, A. Tulloch, Y. Jia and K. He. Accurate, large minibatch sgd: Training imagenet in 1 hour. *arXiv preprint arXiv:1706.02677* (2017).

[11] R. Hadsell, S. Chopra and Y. LeCun. Dimensionality reduction by learning an invariant mapping. *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06)*. Vol. 2. IEEE. 2006, 1735–1742.

[12] K. He, X. Zhang, S. Ren and J. Sun. Deep residual learning for image recognition. *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, 770–778.

[13] A. Hermans, L. Beyer and B. Leibe. In defense of the triplet loss for person re-identification. *arXiv preprint arXiv:1703.07737* (2017).

[14] K. Hornik. Approximation capabilities of multilayer feedforward networks. *Neural networks* 4.2 (1991), 251–257.

[15] G. Huang, Z. Liu, L. Van Der Maaten and K. Q. Weinberger. Densely connected convolutional networks. *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, 4700–4708.

[16] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167* (2015).

[17] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).

[18] A. Krizhevsky, G. Hinton et al. *Learning multiple layers of features from tiny images*. Tech. rep. Citeseer, 2009.

[19] R. Kumar, E. Weill, F. Aghdasi and P. Sriram. Vehicle re-identification: an efficient baseline using triplet embedding. *arXiv preprint arXiv:1901.01015* (2019).

[20] Y. LeCun, L. Bottou, Y. Bengio, P. Haffner et al. Gradient-based learning applied to document recognition. *Proceedings of the IEEE* 86.11 (1998), 2278–2324.

[21] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár and C. L. Zitnick. Microsoft coco: Common objects in context. *European conference on computer vision*. Springer. 2014, 740–755.

[22] L. Liu, H. Jiang, P. He, W. Chen, X. Liu, J. Gao and J. Han. On the variance of the adaptive learning rate and beyond. *arXiv preprint arXiv:1908.03265* (2019).

[23] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu and A. C. Berg. Ssd: Single shot multibox detector. *European conference on computer vision*. Springer. 2016, 21–37.

[24] D. G. Lowe. Distinctive image features from scale-invariant keypoints. *International journal of computer vision* 60.2 (2004), 91–110.

[25] H. Luo, Y. Gu, X. Liao, S. Lai and W. Jiang. Bag of tricks and a strong baseline for deep person re-identification. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*. 2019, 0.

[26] T. M. Mitchell. *Machine Learning*. New York: McGraw-Hill, 1997. ISBN: 978-0-07-042807-2.

[27] V. Nair and G. E. Hinton. Rectified linear units improve restricted boltzmann machines. *Proceedings of the 27th international conference on machine learning (ICML-10)*. 2010, 807–814.

[28] A. Neubeck and L. Van Gool. Efficient non-maximum suppression. *18th International Conference on Pattern Recognition (ICPR'06)*. Vol. 3. IEEE. 2006, 850–855.

[29] H. Oh Song, Y. Xiang, S. Jegelka and S. Savarese. Deep metric learning via lifted structured feature embedding. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2016, 4004–4012.

[30] J. Redmon, S. Divvala, R. Girshick and A. Farhadi. You only look once: Unified, real-time object detection. *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, 779–788.

[31] S. Ren, K. He, R. Girshick and J. Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. *Advances in neural information processing systems*. 2015, 91–99.

[32] E. Ristani, F. Solera, R. Zou, R. Cucchiara and C. Tomasi. Performance measures and a data set for multi-target, multi-camera tracking. *European Conference on Computer Vision*. Springer. 2016, 17–35.

[33] F. Schroff, D. Kalenichenko and J. Philbin. Facenet: A unified embedding for face recognition and clustering. *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015, 815–823.

[34] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556* (2014).

[35] C. Su, J. Li, S. Zhang, J. Xing, W. Gao and Q. Tian. Pose-driven deep convolutional model for person re-identification. *Proceedings of the IEEE International Conference on Computer Vision*. 2017, 3960–3969.

[36] Y. Sun, L. Zheng, Y. Yang, Q. Tian and S. Wang. Beyond part models: Person retrieval with refined part pooling (and a strong convolutional baseline). *Proceedings of the European Conference on Computer Vision (ECCV)*. 2018, 480–496.

[37] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke and A. Rabinovich. Going deeper with convolutions. *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015, 1–9.

[38] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens and Z. Wojna. Rethinking the inception architecture for computer vision. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2016, 2818–2826.

[39] J. R. R. Uijlings, K. E. A. Van De Sande, T. Gevers and A. W. M. Smeulders. Selective search for object recognition. *International journal of computer vision* 104.2 (2013), 154–171.

[40] X. Wang, Y. Hua, E. Kodirov, G. Hu, R. Garnier and N. M. Robertson. Ranked List Loss for Deep Metric Learning. *arXiv preprint arXiv:1903.03238* (2019).

[41] L. Wei, S. Zhang, W. Gao and Q. Tian. Person transfer gan to bridge domain gap for person re-identification. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2018, 79–88.

[42] L. Wei, S. Zhang, H. Yao, W. Gao and Q. Tian. Glad: Global-local-alignment descriptor for pedestrian retrieval. *Proceedings of the 25th ACM international conference on Multimedia*. ACM. 2017, 420–428.

[43] F. Xiong, Y. Xiao, Z. Cao, K. Gong, Z. Fang and J. T. Zhou. Towards good practices on building effective CNN baseline model for person re-identification. *arXiv preprint arXiv:1807.11042* (2018).

[44] L. Zheng, L. Shen, L. Tian, S. Wang, J. Wang and Q. Tian. Scalable person re-identification: A benchmark. *Proceedings of the IEEE international conference on computer vision*. 2015, 1116–1124.

[45] Z. Zheng, X. Yang, Z. Yu, L. Zheng, Y. Yang and J. Kautz. Joint discriminative and generative learning for person re-identification. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2019, 2138–2147.

[46] Z. Zheng, L. Zheng and Y. Yang. Unlabeled samples generated by gan improve the person re-identification baseline in vitro. *Proceedings of the IEEE International Conference on Computer Vision*. 2017, 3754–3762.

[47] Z. Zheng, L. Zheng and Y. Yang. A discriminatively learned CNN embedding for person reidentification. *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM)* 14.1 (2018), 13.

[48] Z. Zheng, L. Zheng and Y. Yang. Pedestrian alignment network for large-scale person re-identification. *IEEE Transactions on Circuits and Systems for Video Technology* (2018).

[49] Z. Zhong, L. Zheng, G. Kang, S. Li and Y. Yang. Random erasing data augmentation. *arXiv preprint arXiv:1708.04896* (2017).

[50] Z. Zhong, L. Zheng, Z. Zheng, S. Li and Y. Yang. Camera style adaptation for person re-identification. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2018, 5157–5166.