Self-supervised Video Representation Learning by Exploiting Playing Speediness Changes

by

Lizhe Chen

A thesis
presented to the University Of Waterloo
in fulfilment of the
thesis requirement for the degree of
Master of Applied Science
in
Computer Science

Waterloo, Ontario, Canada, 2022

Author's Declaration

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

In recent research, the self-supervised video representation learning methods have achieved improvement by exploring video's temporal properties, such as playing speeds and temporal order. These works inspire us to exploit a new artificial supervision signal for self-supervised representation learning: the change of video playing speed. Specifically, we formulate two novel speediness-related pretext tasks, i.e. speediness change classification and speediness change localization, that jointly supervise a shared backbone for video representation learning. This self-supervision approach solves the tasks altogether and encourages the backbone network to learn local and long-ranged motion and context representations. It outperforms prior arts on multiple downstream tasks, such as action recognition, video retrieval, and action localization.

Acknowledgement

I want to thank all the people who made this thesis possible. My supervisor, Professor Olga Veksler, provided many valuable suggestions when I wrote this thesis, including recommending literature worth referencing and guiding me in designing my experiments. Thank you, Olga, for the help in the past two years, and I really enjoy working with you. I also would like to thank the manager of my Master Co-op, Peng Dai, who allowed me to use Huawei Canada's cloud computing services and servers to conduct the experiments.

Table of Contents

Chapter 1

Introduction

With the advent of convolutional neural networks and other machine learning technologies [44, 63, 32, 68, 87], the field of computer vision has achieved rapid development in recent years. Traditional problems involving still images, for example, semantic segmentation [70, 50, 77], object recognition [73, 15, 33] can be solved by end-to-end machine learning methods with good performance. As a result, engineers have created countless applications [64, 23, 36, 83, 92] that bring convenience to our daily lives by combining neural network algorithms with modern hardware. For example, face recognition technology replaces manual password input on our cellphone or realizes automatic driving by combining object recognition and depth detection technology. Thanks to the modern fast GPU computing [57] and new network architectures such as separable convolutional layers and knowledge distillation [35, 34, 28, 27] , the inference processes are significantly boosted to real-time and satisfy the engineering requirements.

There are many applications in computer vision field that require as an input a sequence of still images, that is videos, as opposed to a single image. For example, if our task is to identify an object's action, then the information in the temporal dimension is beneficial. For example, yoga is a popular physical exercise with slow movement, and it will be easier to distinguish it from other actions if the network learns the pace. Learning of pace requires the input to be a video, as opposed to a single image.

Researchers are working towards neural networks can solve complex video-based tasks such as human action recognition [65, 41]. Human action recognition is essential for multiple applications including human-computer interaction [14], patients behaviour monitoring [21, 25], automated surveillance [7, 74], video summarization and labelling [5, 90, 6, 16]. In automated surveillance, for example, the monitoring system of the production line can alert the workers if the cameras detect irregular actions to avoid safety hazards. Teaching computers to understand human motion can also expand the human-computer interaction methods.

More and more TV manufacturers introduced their smart TV with motion-sensing cameras. The sensors can detect human activities, so the users do not need a traditional TV remote anymore. Instead, they can change the TV volumes or pause the video through simple hand movements.

Figure 1.1: An image from Samsung product news. Users can control the new Samsung smart TV with hand movements.

Therefore, the input to many tasks in computer vision (action recognition, list some others as well) is a sequence of image frames, and a neural network architecture has to be modified to adapt to the temporal dimension of the input. In 2013, Ji et al. [38] proposed to use a three-dimensional convolutional network for action classification without any processing on the video. Two papers on the new model architecture published in 2014 [39, 71] played a key role in video processing related research, and several model designs that will be introduced later are extended from these two methods. Researchers have developed many approaches to dealing with video input, but there are still two problems: lack of annotated video data and long training time. One approach to solve these two problems is self-supervised video representation learning.

Representation learning is a concept inherited from traditional computer vision methods. The workflow of traditional computer vision algorithms can be summarized as following steps. First, perform feature extraction on the areas of interest in the image (or video) and use fixed-size vectors to represent them. Then choose a linear classifier [3] based on the extracted representation, and perform the training and prediction. One of the factors in deep learning becoming popular in the early days is simplifying the traditional pipelines to enable end-to-end learning. The neural network learns all the intermediates between the inputs data and final outputs and all steps are trained simultaneously instead of sequentially.

However, when researchers try to deploy deep learning algorithms into tasks in specific fields [61], they soon encounter the problem of insufficient data. Therefore, researchers introduce deep video representation learning by borrowing the idea from traditional algorithms, but now the representation vectors become the intermediate feature layer output of the model. First, we need to design a task named pretext task, so that the labels for this tasks are easy to generate automatically, without the need for human annotation. These automatically generated labels will be used to train network for the pretext task [54, 52]. Since video comprises a series of semantically related frames, it has seful spatial and temporal features. The temporality of these frames implies specific inference rules and physical logic: for example, the movement of objects should be smooth, and the gravity of the earth should be downward. Our hypothesis is that even if the pretext task has no practical application [31], the neural network can complete this task only by learning a useful spatial-temporal representation in its feature layers. Then we can transfer the learned representation to other meaningful tasks like action classification through sharing the weights of the features layers and finetuning with annotated data.

In summary, video representation learning has two applications. The main application is to enhance neural network performance and reduce the training time by transferring knowledge from abundant unannotated data to neural networks when we have insufficient annotated video data for training. Many papers [30, 8, 81], prove that a good pretext task can help video representation learning to obtain reliable data transfer abilities. Secondly, the representation network can help us compare and calculate the similarity among two different video clips by calculating their representation vectors' similarity score (e.g. dot product) [8]. Usually, we can consider two video clips similar if they contain the same actions in different environments. The same actions should have similar spatial-temporal features and the feature representation network should output close feature vectors for these action. Calculating the similarity between two video clips helps us perform video retrieval [49, 8] (searching similar video clips from other videos rather than the query video) and temporal action localization [8, 24] (finding the occurring time of the query action in a single long video). The former task can be used for a video recommendation system [9], and the latter task can help us to summarize the video contents [85].

1.1 Video Representation Learning

Representation learning is essential in multiple computer vision tasks such as object recognition, 3D model reconstruction, or robot navigation. Typically, we want the machine to learn a representation that includes crucial information related to the input context. But there is no restriction on feature format, and the definition usually depends on the type of problem or method. The traditional hand-crafted image feature descriptors such as SIFT

[51] define the features as small image patches that help identify the objects. The image features are usually computed from some regions with interest points like corners, edges or ridges. Similarly, researchers also proposed many hand-crafted spatio-temporal descriptors for video representation, such as histograms of oriented 3D Spatio-temporal gradients to build HOG3D descriptor [43] or looking for Spatial-temporal interest points (STIP [46]). In 2013, Wang et al. [79] proposed improving dense trajectories (IDT) descriptors, which achieved the best results among all hand-crafted video representation learning approaches. However, hand-crafted methods have significant drawbacks. First, selecting which features are valid for the target task is necessary, which means engineers have to decide or filter features through a lengthy trial and error process. Second, we might also need to manually tune a large group of parameters in the algorithm to get the best results. As the difficulty of the task increases, the cost of the selection and finetuning can be prohibitive.

Deep learning is successful in computer vision to a large degree because intermediate engineering feature design steps are down automatically with neural networks. While most neural networks were designed for still images, there is increasing research in designing neural networks for video tasks. The most obvious way to take sequential image data is by changing the 2D convolutional layers to 3D. Many well-established networks for video representation learning inherit the designs of some popular image processing neural networks such as C3D [71] (based on VGG network [63]) and I3D [10] (based on Google Inception network [67]). The transformer is a widespread technique in the natural language processing field, and since we can consider the video inputs as sequences, researchers have also tried to build video transformers. For example, Arnab et al. from the Google research introduced their ViViT [6] video transformer recently, and the performance is comparable to the previous convolutional neural network approaches.

Unlike the traditional methods, the deep learning approach trains the neural networks with many image samples to discover the latent patterns in images and automatically work out the most defining features. Although the trade-offs are higher computing requirements and longer training time, modern engineers prefer the deep learning approach, which performs far better than traditional methods. Also, the development of convolutional neural networks has had a significant influence in recent decades. This bursting trend has been enabled by a big improvement in GPU computing power and an increased amount of data available for training neural networks.

The main problem of all supervised video representation learning is the lack of annotated video data. We need enough annotated data to apply supervised learning. However, labelling video data is time-consuming and expensive. Compared to image annotation, the burden of processing video data is much heavier because the annotators have to pay close attention to the timeline if they want to crop out the target action from the raw videos. Furthermore,

sometimes it isn't easy to obtain data in certain areas. For example, patient data is often protected by law in healthcare, making it difficult to collect related data such as patient's video recording. The large number of network parameters also cause a long training time. Take training C3D [71] with Tesla K40 GPUs as an example; the training set UCF101 [65] is a relatively small action recognition dataset with 9K videos but the training time for one epoch is more than 19 hours when the input dimension is 16 frames of 256x256. By using a network pretrained on a pretext task for multiple downstream applications, we can save computational time.

1.2 Self-supervised Methods

Self-supervised video representation learning teaches neural networks to extract helpful video features from the abundant un-annotated videos by performing a particular task that doesn't need manual annotation work. Instead, the data samples are transformed or sampled in order to automatically obtain labels for some task which is not the final task of interest. This task is named the pretext task because our aim is not developing a classifier for this self supervised task. Instead, we want our network to learn an efficient latent representation of the videos so that we can use this video representation in the network training with the downstream tasks, where the downstream task is actually the task we are interested in. Downstream tasks, such as action recognition, are typically more challenging than the pretext tasks, and require user annotation effort to construct ground truth. A good pre-trained network should help us extract useful video features from the original videos and reduce the training time significantly. Thus, the evaluation of the self-supervised network is based on how well we can fine tuned it using annotated datasets to perform one or more downstream tasks.

Fully supervised training is more straightforward but challenging to guarantee performance and versatility if the training dataset is small. The annotation work for video-based tasks like action recognition is much heavier than the traditional image labelling because the workers need to watch the whole video for each sample to annotate all query actions' time boundaries accurately. In the industry, acquiring high-quality annotated video data is very time-consuming and costly. However, the unannotated videos are relatively easy to collect through the internet. The main benefit of self-supervised learning is unbounded video datasets at the pre-train stage. Pre-training with unannotated but related videos can help our networks learn critical features, thus increasing the training efficiency of the downstream tasks. Self-supervised training is also an excellent solution for industry projects when developers don't have much-annotated data at the start time. This strategy can help them make an early delivery while still waiting for more annotated data.

We can describe self-supervised video representation learning as a two-stage training strategy

to solve complicated video-based tasks like human action recognition. The first stage will be self-supervised training the network through a pre-designed task that applies appropriate transformation on the original video to automatically generate an effective supervisory signal based on the transform itself or certain constraints(i.e. the pretext task). Then, we can freely take advantage of billions of video data from any source. In the second stage, we will evaluate the video representation learning performance by finetuning the pre-trained network from the first stage for a specific video-related job such as action recognition, video retrieval (i.e. downstream task). Of course, we still need an annotated dataset during the finetuning. However, this two-stage training routine benefits us to reuse the pre-trained network for different downstream tasks. Also, compared to fully supervised training using the annotated dataset, starting from the self-supervised pre-trained network will reduce the converge time significantly.

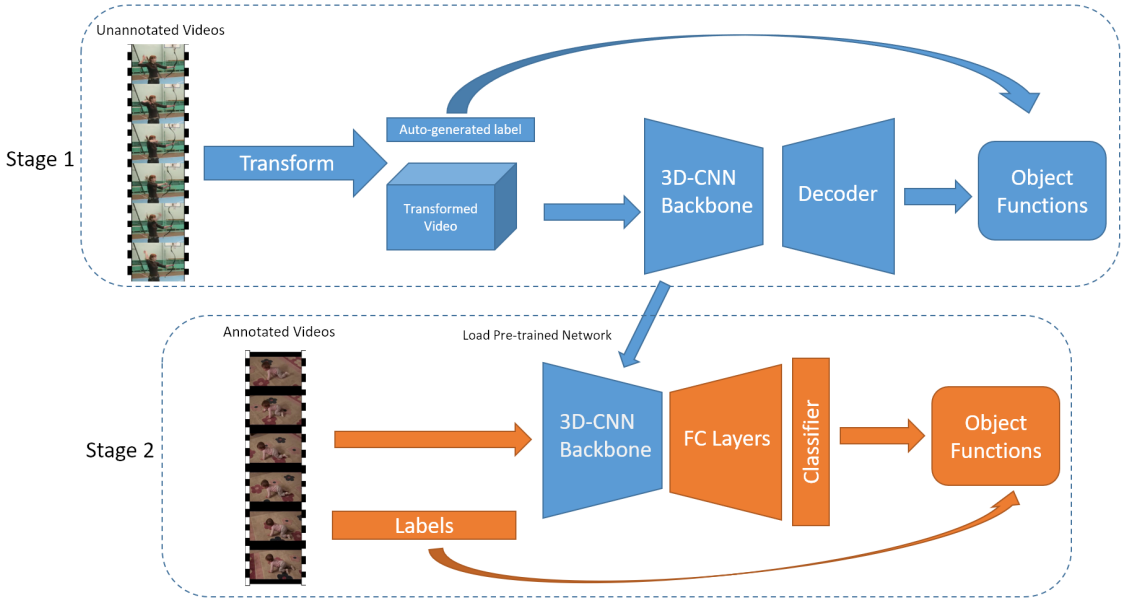


Figure 1.2: An image of two stage training

1.2.1 Pretext Tasks

As shown in Figure1.2, we compose the network in two parts: a encoder or backbone part, a convolutional neural network used to transform the input video clip into a latent feature vector; and a decoder part which uses the feature vector to do the inference task such as classification or regression. At the first stage, we want our network to learn the video representation of unannotated data through a specific simple task named pretext task. The core requirement for the pretext task is that for each video sample, one can generate target

labels automatically. Also, the learned representation should be easily adaptable for other tasks unknown during training for the pretext task. For example, we can ask the network to classify whether the input video clip (a sequence of video frames) is in order or shuffled [54, 84]. Each video sample contains a multiple of frames. We can sample three frames, and if they are extracted in continuous order, we label it as positive; otherwise, if they are shuffled, the label is negative. Then, we can program our data sampler to shuffle the video clip randomly and generate a binary label on the fly. The network can only complete order classification tasks when it understands the underlying video content and learns representative Spatio-temporal features. Therefore, after the pretraining with such a pretext task, we can use the encoder weights of the network as the initial weights for other tasks like action recognition.

1.2.2 Downstream Tasks

Usually, the downstream tasks are applications uses to evaluate the quality of features learned by self-supervised learning. The downstream task can be a classification or detection task and can benefit from the pretrained models when annotated training data are insufficient. For video representation learning, the most common downstream task is action recognition or video classification. Another usual downstream task is video retrieval: given a query video and a set of candidate videos, select the video which corresponds to the query most. Typically, the videos are returned as a ranked list of candidates and scored via retrieval metrics. For the evaluation of pretext task pretraining, the retrieval metric is usually the similarity scores among the feature vectors of query and candidate.

The primary evaluation method of our pretext task design is using the pre-trained network for downstream task training. This type of evaluation is a case of transfer learning. A good pretext task design should learn a video representation that needs a little training. We will build a new classifier on top of the backbone, and the training of downstream tasks includes two approaches: linear classifier and finetuning. We can either freeze all parameters in the pre-trained backbone and only train the small linear classifier to perform the downstream task or finetune all parameters simultaneously but limit the number of training epochs. The finetuning method is more common in self-supervised video representation learning [84, 8, 81]. For example, the usual epoch number for finetuning action recognition tasks is 20-30 epochs for a large dataset like Kinnetics400, or 200-300 epochs for a smaller HMDB51. The performance on downstream tasks can reflect the efficiency of our pretext task design. Usually, a harder pretext task will improve downstream task performance. However, the network may also need a lot of time to converge at the pretraining stage if the pretext task is too hard, making self-supervised learning less useful.

1.3 Contributions

Recently, [8, 81] suggested video speediness classification as an efficient pretext task for video representation learning. This idea was inspired by the fact that the human viewer can often quickly notice if the playing speed of a video jumps up. Sped up a video reduces the playing FPS (frame per second), thus reduces the smoothness of the object's movement. The jiggling effect is disturbing to the viewer because we usually know the motion dynamic of actions or sports. The setting of speediness classification is simple: each data sample is a frame sequence with a fixed length and is extracted from the original video at a random start point for every p frame. The range of integer p is defined, and the network has to classify the value of p. Therefore, the data sampler can generate the input data and target label at the same time. Experiments prove results proved that speediness classification is a reliable pretext task, and Wang et al. [81] claimed they got the best result when defining p within $[1, 4]$.

However, the speediness classification does not use the relationship among the different playing speeds, i.e. the motion dynamic increases as the p increases. For the frame sequences sampled from the same action, same object but different speediness, only the sampling methods cause the differences between the features, not the contexts. The network should focus on these differences because if some features change more than others as the playing speediness changes, these features are more likely to come from the moving object, which is more critical to representation learning. Converting the speediness classification to regression can help reveal the speediness relationship, but it will also unnecessarily increase the hardness of the pretext task [8]. It is hard to quantify the speediness of the frame sequence precisely because the object in the video can also affect the speediness. However, we assumed that if the speediness is changed within the frame sequence, then the features extracted from such sequence must relate to the speediness change. We can call such frame sequence as speediness changing clip and the previous ones as the constant clip. If the network can learn features from both types of sequence, it may help to develop better video representation.

Therefore, we design two different approaches to learn the features of speediness change clips:

(1) The first approach is extending the previous method. On top of the pretext task designed by [81], we add four new classes: two speediness jump-up classes and two speediness drop-down classes. The speediness is increased (jump-up) or decreased (drop-down) at the middle of the sequence in these new classes. And speediness change is further classified by the extent of the change: it can be one-level change (e.g. sampling rate changes from 1 to 2, or 3 to 2) or two-level change (e.g. 2 to 4, 3 to 1). Same to [81], we still have four regular speediness classes start from $p = 1$ to $p = 4$, so there are eight classes in total. The sampling probabilities are uniform for each category, and the pretext task is classifying the input frame sequence into these eight classes.

(2) The second approach is a combination of speediness-change classification and localization. The clip creating method is similar to the first approach, but we randomly select the change point of changing speediness clip instead of fixing it at the middle of the sequence. Thus, the network has two branches of classifiers to perform speediness change classification and localization simultaneously. However, if we use the same settings in the first approach, task may be too difficult and the training may not converge. Therefore, we reduce the number of classes to three by merging the categories with the same speediness change pattern. The network only has to classify the changing pattern as constant, jump-up or drop-down. The speediness-change localization task asks the network to predict the speediness-change position in the sequence. Each constant speediness sample will be mapped to a dummy localization label.

We further apply contrastive learning [12] to regularize the classification for both approaches. We evaluated each pretext task by two downstream tasks: action recognition and video retrieval on HMDB51 [45] and UCF101 [65] datasets. We also did the ablation study on both approaches to assess each sub-component in pretext tasks. Our experiments show that both approaches outperform the previous methods on modern video representation networks, and the second multi-task approach is even better than the first one.

1.4 Thesis Organization

In the next chapter, we will further introduce some basic concepts about modern convolutional neural networks, self-supervised learning, and contrastive learning. Chapter 2 also introduces more previous self-supervised video representation learning methods, and some popular neural network architecture for video representation learning including the models we used for this work.

Chapter 3 describes our first pretext task: speediness change classification. Chapter 4 describes our multi-task approach, which combines the classification and localization task of speediness change. The experiment details and results of both methods are in Chapters 3 and 4, respectively. Chapter 5 summarizes and concludes our work in this thesis and proposes some possible future research directions.

Chapter 2

Related Work

2.1 Convolutional Neural Network

The convolutional neural network (CNN) is a feed-forward neural network. Its artificial neurons can respond to the neighbour units in the coverage area. Therefore, it has excellent performance for image processing. The convolutional neural network includes convolutional layers with corresponding pooling layers for learning the features. Also, we need a few fully connected layers to perform the final classification or regression task [47]. This design enables convolutional neural networks to extract the two-dimensional spatial features of the input data. Compared with other deep learning structures, convolutional neural networks can give better performance on image-related tasks. We can use back-propagation algorithms to train this type of network. Furthermore, convolutional neural networks need to consider fewer parameters than other feed-forward neural networks, making it an attractive deep learning structure.

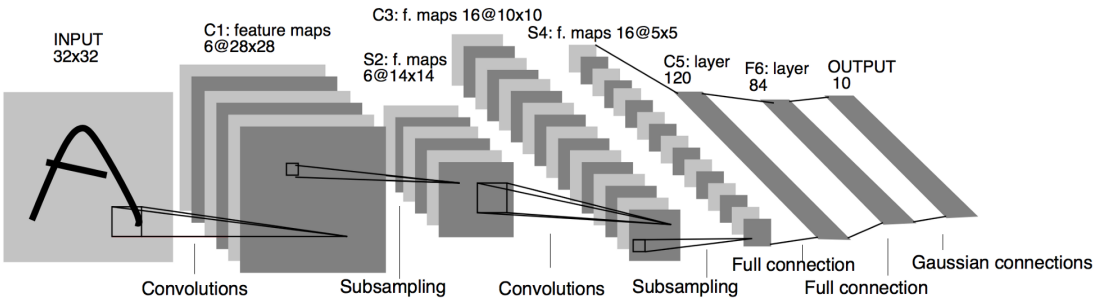


Figure 2.1: LeNet-5 architecture as published in the original paper [47].

In 1989, Yann LeCun et al. published a convolutional neural network structure named LeNet [47]. From the current point of view, LeNet is a small network. However, LeNet is

a milestone in the CNN network structure, defining the modern CNN network structure for the first time and demonstrating its application in image classification. The paper defines the basic framework of Convolutional Neural Network: Convolutional Layer + Pooling Layer + Fully Connected Layer. Compared with the fully connected layer, the convolutional layer has two advantages: local connection and weight sharing. When the human eye observes the outside world, it first observes the local information of the object and then obtains the global knowledge through the local feature, that is, recognizes what the object is. So if we design a neural network according to this principle, every neuron does not need to perceive the global image. Instead, each neuron only perceives the local image area. Then, the network can obtain comprehensive information at a higher level by synthesizing these neurons that sense different parts. First, the convolutional layer uses convolution to achieve local connections. Then each neuron in the output data convolves the image with the same convolution kernel (shared weight) and adds the same bias. In the classic fully connected layer, a neuron needs to correspond to a weight and a bias. But in convolutional layers, each neuron corresponds to the same convolution kernel and the same bias, significantly reducing the parameter amount.

The operation performed in the convolutional layer is called convolution. Each convolution operation multiplies each element on the convolution kernel with the corresponding input data (pixels), adds all the results, and then moves the convolution kernel by one stride until the convolution kernel traverses the input data. Sometimes, we want to pad the input matrix with fixed data (such as zero) before the convolution operation. Padding helps the network to keep information at the borders and preserve the height and width of the feature map. It allows us to design deeper networks. Without padding, reduction in volume size would reduce too quickly. We can call the input and output data of the convolutional layer a feature map. In the convolutional layer, each convolution kernel extracts a feature, so the number of convolution kernels is equal to the number of output feature maps. To calculate the height and width of output feature map (C, H, W), assume we have input size (c, h, w), C kernels with size (c, kh, kw), padding p, stride s:

$$(H, W) = \frac{(h, w) + 2p - (kh, kw)}{s} + 1 \tag{2.1}$$

The pooling layer usually follows a convolutional layer to reduce the data size. The pooling layer also has a pooling window to move the feature map. Pooling operation calculates the maximum or average value in the pooling window in each move, so the pooling layer typically has no parameters to learn. The pooling layer sub-samples the image to retain helpful information and to be robust to small position changes. When the input data has a slight deviation, the pooling will still return the same result.

In 2012, Krizhevsky and Hinton launched AlexNet. They won the ILSVRC (ImageNet Large-Scale Visual Recognition Challenge) that year with an absolute advantage of 10.9 percentage points above second place, which aroused many scholars' research on deep learning. At that time, the computing power of AlexNet is not as strong as it is now. The GTX 580 used by AlexNet has only 3GB of video memory, but it has completed a significant breakthrough on ImageNet [15]. AlexNet uses a dual GPU framework. Many of the current deep learning frameworks have this ability. Still, at the time, it was only possible to design a larger and deeper network by manually implementing the underlying code. AlexNet uses ReLU instead of Tanh as the activation function to solve the problem of gradient disappearance and speed up the network convergence. The stride in AlexNet's pooling operation is smaller than the pooling kernel size, so adjacent pooling areas have overlapping parts. This operation is called overlapping pooling. According to the author, this operation can reduce the index top-1/top-5 error rate by 0.4%/0.3% and reduce over-fitting. In terms of data enhancement, the authors randomly crop the training image from 256 $\times$ 256 to 224 $\times$ 224. They cut image patches from the four corners and the center of the image during the test and performed mirror flipping. They collect ten patches and average the results of these patches to calculate the final prediction result. Due to the continuous development of GPU power, AlexNet appears too "small" for many tasks. However, the above improvements made by the author are worthy of reference. For example, when training a model and the structural design of the model itself, we can use additional training techniques such as data reinforcement and dropout to improve the efficiency of the training model.

In 2014, Google proposed the Inception network and built GoogLeNet [68] based on it. Inception "deepened" the network by building Conv Layer Groups (or blocks), which combine multiple convolutional layers plus activation function. The author pointed out that the most direct way to improve the expressive ability of the model is to increase the "size" of the model, which in turn will lead to two problems: the larger the model, the larger is the number of network parameters, and the easier it is to overfit and to require a larger dataset, but the construction cost of a large dataset is very high; the larger the model, the greater the demand for computing resources, which is unacceptable in practical tasks. The author believes that the primary method to solve these two problems is changing the fully connected and convolutional layers to a sparse network structure. The Inception module contains parallel convolutional layers with kernel sizes of 1$\times$1, 3$\times$3, 5$\times$5 and pooling layers. Then they concatenate the feature maps obtained by the subsequent input. The author hopes that the deep Inception Module in the model can capture higher abstraction. The density of the convolutional layer in the Inception is gradually reduced to capture features of a larger area. Therefore, the deeper the Inception, the more the output channels of the convolutional layers. But this will increase the amount of calculation significantly. So the authors suggests deploying the convolutional layers with a kernel size of 1$\times$1. These layers can

increase non-linearity to improve the network's expressive ability and simultaneously reduce dimensionality, decreasing the calculation cost. Since the Inception Module maintains the same input and output size, GoogLeNet can be modularized according to task requirements.

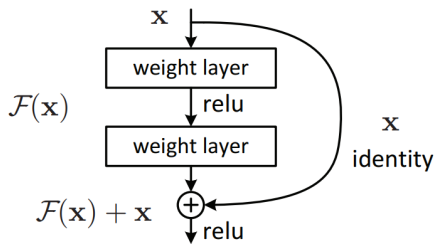


Figure 2.2: Residual learning: a building block [32]

In 2015, Kaiming He proposed ResNet [32] to solve the degradation problem in neural networks (i.e. the problem that with more layers, CNNS have a larger training/test errors). Furthermore, the proposed residual block significantly improves the learning ability of neural networks. The author presents residual learning to solve the degradation problem. For a stacked-layer structure (Conv Layer Group), we define the input as x, the learned feature as $H(x)$, and now we hope it can learn the residual $F(x) = H(x) - x$ so that the original learning feature becomes $F(x) + x$. The reason for this is that residual learning is easier than learning original features directly. When the residual is 0, the layer group only performs identity mapping at this time; at least the network performance will not decrease. In fact, the residual will not be 0, making the layer group learn new features based on the input features to have better performance.

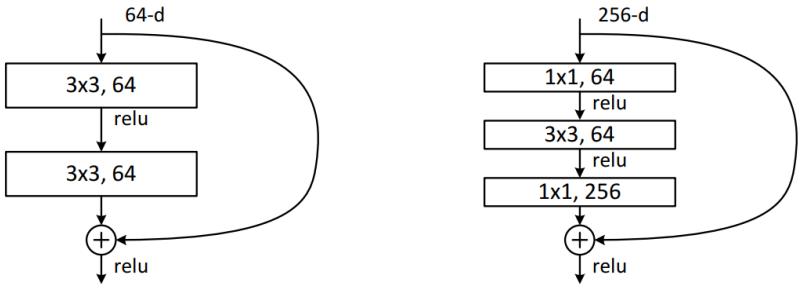


Figure 2.3: A deeper residual function F for ImageNet. Left: a building block (on 56×56 feature maps) for ResNet34. Right: a "bottleneck" building block for ResNet-50/101/152. [32]

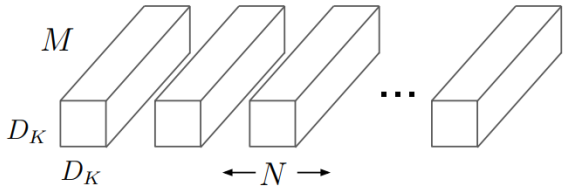
Since the residual block does not require additional parameters and calculations, Kaiming conducted multiple sets of controlled experiments to prove the effectiveness of the network structure. However, suppose the depth of the model is to continue to increase. In that case, it needs to be improved: the original residual block (shown on the right side of the figure 2.3) is enhanced to bottleneck block to reduce the parameters and calculations of the model. ResNet is modified based on the VGG network [63] and added residual learning. For 18-layer and 34-layer ResNet, it performs residual learning between two layers. When the network is deeper, it performs residual learning between three layers.

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

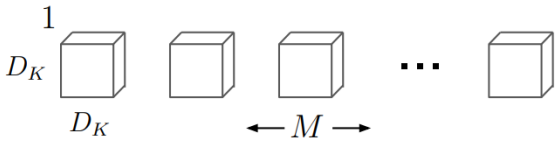
Figure 2.4: A table of architectures for ImageNet. Building blocks are shown in brackets, with the numbers of blocks stacked. Downsampling is performed by conv3_1, conv4_1, and conv5_1 with a stride of 2 [32]

Around 2017, researchers became interested in lightweight neural networks, i.e. networks with less parameters. Standard methods include designing lightweight network models or compressing trained complex networks (such as pruning, etc.). At this time, Google proposed a lightweight model: MobileNet [35], which became the baseline model that people often use for comparison. In MobileNet, the authors use depthwise separable Convolution to design a lightweight network. Furthermore, users can change the network width (channels) and input resolution according to their needs by setting the two hyperparameters: width multiplier and Resolution Multiplier. Thus, the users can trade off the model between latency and accuracy according to the actual scene.

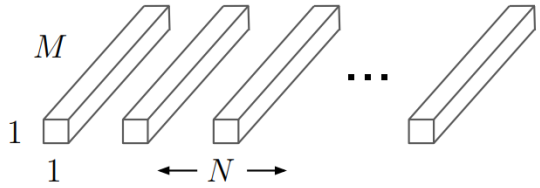
As shown in Fig 2.5, depthwise separable Convolution splits standard convolution into two parts: depthwise convolution and pointwise convolution. The shape of each filter in the original standard convolution is (D_k, D_k, M), where M is the channel number of the input feature map. Each filter performs convolution on the input feature map, and the output shape is $(D_out, D_out, 1)$. If there are N filters, the output Shape is (D_out, D_out, N). On the other hand, in the depthwise convolution stage, the shape of each depthwise convolution fil-



(a) Standard Convolution Filters



(b) Depthwise Convolutional Filters



(c) 1×1 Convolutional Filters called Pointwise Convolution in the context of Depthwise Separable Convolution

Figure 2.5: The standard convolutional filters in (a) are replaced by two layers: depthwise convolution in (b) and pointwise convolution in (c) to build a depthwise separable filter. [35]

ter is $(D_k, D_k, 1)$, and there are M filters in total. Therefore, the shape of the output result is (D_out, D_out, M). Next, in the pointwise convolution stage, each pointwise convolution filter has the shape $(1, 1, M)$ and N filters in total. Thus, the shape of the output result is (D_out, D_out, N). Accordingly, we can calculate that the required parameter of standard convolution is $D_k \times D_k \times M \times N$ and the required parameter of depthwise separable convolution is $D_k \times D_k \times M + M \times N = M * (D_k \times D_k + N$. Compared with standard convolution, the reduced number of parameters is $N - 1 \times D_k \times D_k - N$.

2.2 Self-supervised Learning for Images

Self-supervised learning allows us to derive various labels from data without additional cost. This motivation is very straightforward. The cost of generating a data set with "clean" (no noise) labels is high, but unlabeled data is produced all the time. In order to use a large amount of unlabeled data, one solution is to set a reasonable learning goal to obtain the supervision signal from the target data itself. Self-supervised tasks (also called pretext tasks) allow us to deploy the supervision loss function. Most of the time, are not interested in the final performance of the pretext task. We are only interested in the learnt latent representations. We hope that these representations can cover good semantic or structural meanings and benefit various downstream practical tasks. For example, we can randomly rotate images and train a model to predict how each input image is rotated. This rotation prediction task is artificially constructed, so the actual accuracy is not essential, just like we treat auxiliary tasks. However, we expect the model to learn high-quality latent variables for real-world tasks using a small amount of labelled data. Actually, all generative models can be regarded as self-supervised. However, their goal is different: generative models focus on creating a variety of realistic pictures, while self-supervised representation learning focuses on generating good features that are generally helpful for multiple tasks. Researchers have put forward many ideas for self-supervised representation learning of images. The common workflow is to train a model on one or more pretext tasks that use unlabeled images and then use an intermediate feature layer of the model to provide input for the multi-class logistic regression classifier of the ImageNet classification task or other downstream tasks. Some researchers have recently proposed using labelled data to train supervised learning while using unlabeled data to train self-supervised pretext tasks with shared weights [88, 66].

The first type of pretext tasks is image transformation recognition. We expect that subtle distortions on the image will not change its original semantics or geometric form. We can think that the slightly deformed image is the same as the original image, so it is expected that the learned features are invariant to the deformation operation. Dosovitskiy et al. published in 2015 "Exemplar-CNN" [19] using unlabeled image patches to create an alternative dataset. They sampled images at different locations and different scales to obtain

N 32*32 patches. They only select patches from areas with large gradients because these areas contain edges and are more likely to contain objects or parts of objects. After obtaining the base patches, they deform each patch by applying various random transformations (translation, rotation, scaling, etc.). The deformed patches obtained from the same initial patch are regarded as belonging to the same class. Their pretext task is to teach the network to distinguish between the different classes. We can create as many classes as we need and do not require manual annotation.

Figure 2.6: left: Exemplary patches sampled from the unlabeled dataset. Right: In the upper left corner is the original patch of a deer. After applying the random transformation, various deformed patches are produced. In the pretext task, all these tiles should be classified into the same category. [19]

Rotating the entire image is another low-cost method to modify the input image while keeping the semantic content unchanged. Each input image is first randomly rotated by multiples of 90 degrees, corresponding to [0°, 90°, 180°, 270°]. Then, Gidaris et al. [22] train the CNN model to predict the rotation class, which is a 4-classification problem. In order to recognize the same image rotated by different angles, the model must learn to recognize high-level target parts (such as the head, nose, and eyes) and recognize the relative positions of these parts instead of only recognizing local patterns. The pretext task enables the model to learn the semantic concepts of objects in this way.

The second type of self-supervised learning task extracts multiple patches from an image and requires learning the position relationship between these patches. A paper published by Doersch et al. in 2015 [17] formalized the pretext task as predicting the relative position between two random patches in the same image. In order to recognize the relative positions of different parts, the model needs to understand the spatial environment of the target. The patch sampling method used for training is as follows. The first patch is randomly sampled without referring to any image content. Then it is considered that the first patch is in the center of a 3*3 grid, and the second patch is sampled from 8 locations adjacent to the first patch. In order to avoid that the model only captures low-level unimportant signals (for example, connecting a straight line that crosses a boundary or pairing local patterns), they apply additional noise through data augmentation: (1) increase the gap between patches (2) small position jitter (3) randomly down-sample some patches, and then up-sample them to

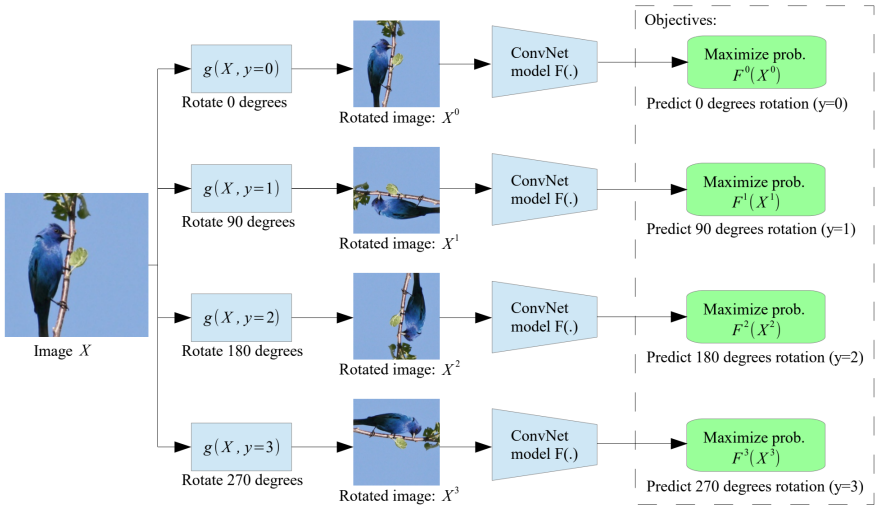


Figure 2.7: Self-supervised learning by rotating the entire image. The model learns to predict how many degrees the picture is rotated. [22]

achieve robustness to pixelation (4) randomly discard two of the three colour channels or convert green and red colour on patches to grey to remove the chromatic aberration, which may cause trivial solution.

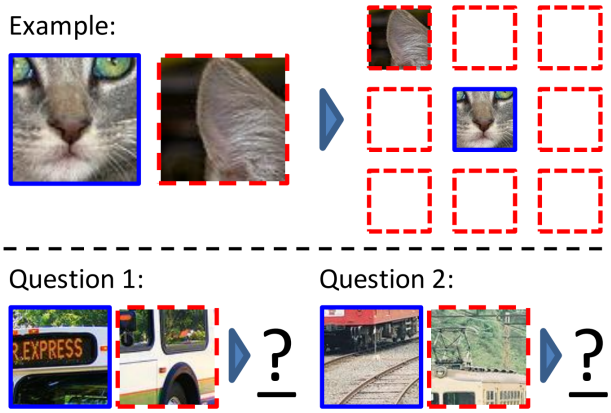


Figure 2.8: Self-supervised learning by predicting the relative position of two random patches

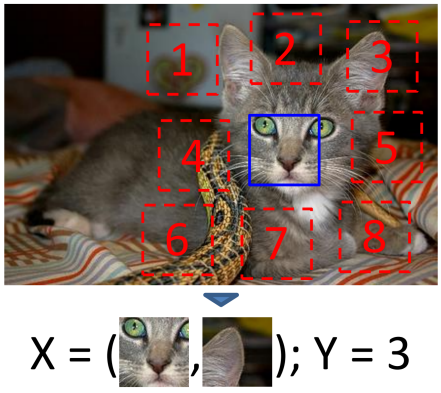


Figure 2.9: The algorithm receives two patches in one of these eight possible spatial arrangements, without any context, and must then classify which configuration was sampled. [17]

Generative modelling as a pretext task is to reconstruct the original input while learning meaningful latent representations. Generative Adversarial Networks (GAN) can learn the mapping from simple latent variables to complex data distributions. Many studies have shown that the latent space of this generative model can capture semantic changes in data.

For example, when we use face data to train GAN, some latent variables are related to features such as facial expressions, glasses, and gender [89]. The denoising autoencoder will learn to restore the original image based on the partially damaged, or random noise image [78]. The inspiration for this design stems from the fact that even if there is noise, humans can easily identify objects in the picture, which shows that the algorithm can extract key visual features and separate them from the noise. Thus, we can train the context encoder to fill in a certain piece of the image [59]. Let $\hat{M}$ be a binary mask, where value 0 means discarding the pixel, and a value of 1 means keeping the input pixel. The shape of the deletion area defined by the mask is arbitrary. We can use a combination of L2 reconstruction loss and adversarial loss to train the model:

$$L(x) = L_{adv}(x) + L_{recon}(x) \tag{2.2}$$

$$L_{recon}(x) = \|(1 - \hat{M}) \odot (x - E(\hat{M} \odot x))\|_2^2 \tag{2.3}$$

$$L_{adv}(x) = \max_D \mathbb{E}_x[logD(x) + log(1 - D(E(\hat{M} \odot x)))] \tag{2.4}$$

where $\odot$ means elementwise product, $D(.)$ is the decoder and $E(.)$ is the encoder as Fig. 2.10

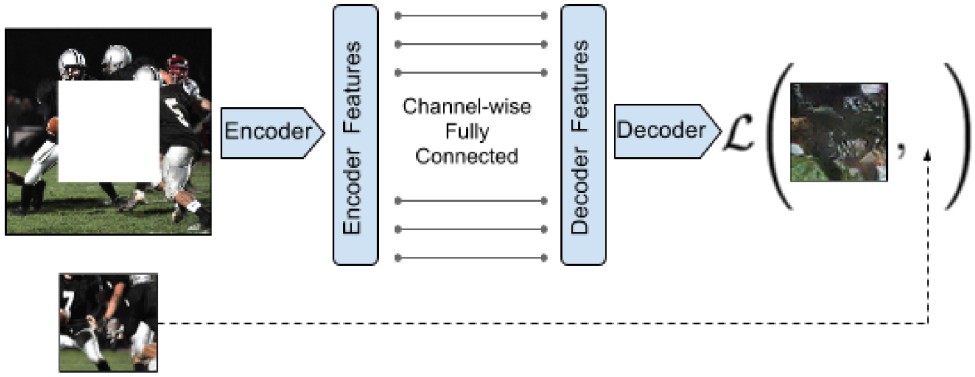


Figure 2.10: Context Encoder. The context image is passed through the encoder to obtain features which are connected to the decoder using channel-wise fully-connected layer. The decoder then produces the missing regions in the image. [59]

2.3 Self-supervised Contrastive Learning

Contrastive self-supervised learning technology is a promising method, which builds representations by learning to encode things that make two things similar or different. Generative self-supervised learning requires the model to reconstruct the image or part of the image.

This type of task is relatively complicated and requires pixel-level reconstruction. The image encoding in the middle must contain much information. Compared with generative self-supervised learning, the task of contrastive learning is less complicated. At present, contrastive learning seems to be in a state of "no clear definition but guiding principles." Its guiding principle is: by automatically constructing similar and dissimilar instances, and it is required to acquire a representation learning model. This model attracts similar instances closer to the projection space, and dissimilar instances are repelled in the far distance in the projection space. The key points are how to construct similar and dissimilar instances and construct a representation learning model structure that can follow the above guidelines. However, the main difficulty of self-supervised contrastive learning is that we do not know which ones are the target class and which ones are other classes. There is no way to determine which pictures should go to maximize similarity and which ones should go to minimize similarity.

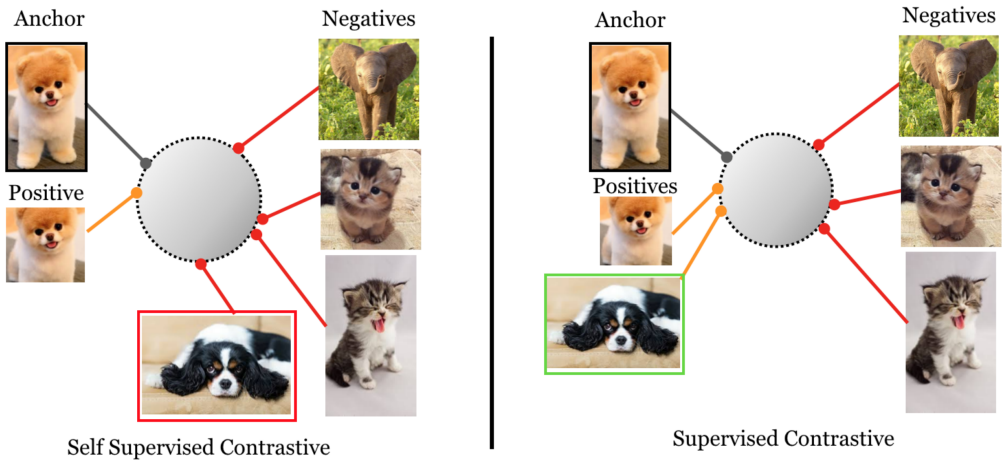


Figure 2.11: Contrastive learning can be used for supervised learning and self-supervised learning. Here we will discuss the self-supervised contrastive learning only [42]

SimCLR [11] is a classic work in the field of self-supervised learning. The SimCLR (4×) model can achieve 76.5% of top 1 accuracy on ImageNet (Fig 2.12), which was 7 points higher than the SOTA model at the time. If this pretrained model is finetuned with 1% of ImageNet labels, SimCLR can reach 85.5% of top 5 accuracy. This is very impressive accuracy with so little supervised information.

As mentioned earlier, contrastive learning is self-supervised learning. We do not label the data, so we need to construct similar data (positive examples) and dissimilar data (negative

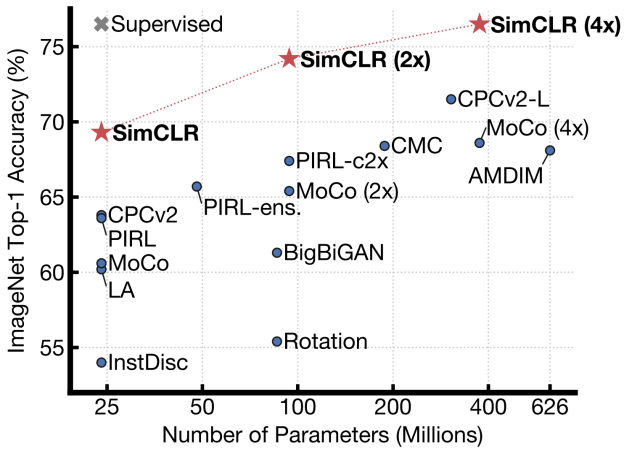


Figure 2.12: CImageNet Top-1 accuracy of linear classifiers trained on representations learned with different self-supervised methods (pretrained on ImageNet). Gray cross indicates supervised ResNet-50. [11]

examples) automatically. For a certain picture, we randomly select two transformations from the set of possible data augmentation operations T (Fig 2.13) : $t \in T$ and $t' \in T$, which act on the original image respectively to form two augmented new images (x, x'). Those are mutually positive examples. During training, any other image in the batch can be used as a negative example of x or x'. In this way, contrastive learning hopes to acquire a certain representation model, which can map the picture to a certain projection space, and narrow the distance of positive examples in this space, and push the distance of negative examples farther. In other words, the representation model is forced to ignore surface factors and learn the internal consistent structural information of the image, that is, to learn certain types of invariance, such as occlusion invariance, rotation invariance, and colour invariance. SimCLR proves that if a variety of image augmentation operations can be integrated simultaneously, the difficulty of the contrastive learning model task will be increased, and the contrastive learning effect will be significantly improved.

SimCLR proposed projection head, which uses a learnable non-linear projection between representation and contrast loss, and the effect is excellent. The advantage of using a learnable network is to avoid the loss of similarity calculation and lose some critical features during training. The paper uses a very simple single-layer MLP with ReLU activation function as a non-linear projection. For an image x_{input} in batch, the corresponding augmented images are x_i and x_j respectively, then the data pair (x_i, x_j) is a positive example, and any other augmented images in batch are negative examples. After $f(g(\cdot))$ transformation, the augmented image is projected into the latent space. We hope that the positive examples are attracted closer, and the negative ones are farther away in the representation space. It is generally achieved by defining a suitable loss function. But we first need a metric function to judge the distance between two vectors in the projection space. Generally, the similarity

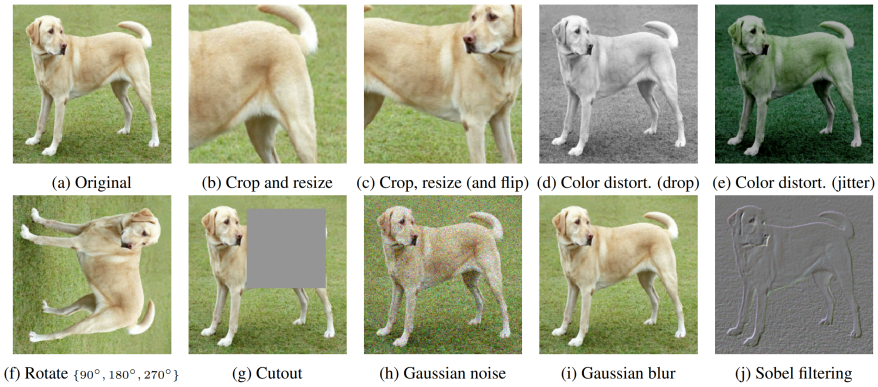


Figure 2.13: Studied augmentation methods in SimCLR paper. [11]

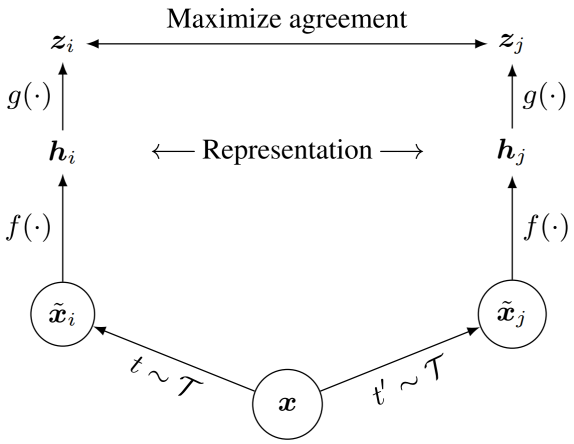


Figure 2.14: A simple framework for contrastive learning of visual representations. Two separate data augmentation operators are sampled from the same family of augmentations ($t \sim T$and $t' \sim T$) and applied to each data example to obtain two correlated views. A base encoder network $f(\cdot)$ and a projection head $g(\cdot)$ are trained to maximize agreement using a contrastive loss. After training is completed, we throw away the projection head $g(\cdot)$ and use encoder $f(\cdot)$ and representation h for downstream tasks. [11]

function is used as the metric. Specifically, the similarity calculation function takes the dot product after the regularization or the Cosine similarity between the vectors:

$$S(z_i, z_j) = \frac{z_i^T z_j}{(\|z_i\|_2 \|z_j\|_2)} \tag{2.5}$$

The object function of SimCLR is InfoNCE loss [75]. The corresponding InfoNCE loss for an input example is:

$$L_{i,j} = -log \frac{(exp(S(z_i, z_j)/\tau)}{\sum_{k=1}^{2N} \mathbf{1}_{[k \neq i]} exp(S(z_i, z_k)/\tau))} \tag{2.6}$$

Where $\mathbf{1}_{[k \neq i]} \in 0, 1$ is an indicator function evaluating to 1 if and only if $k \neq i$ and τ denotes a temperature hyperparameter. (z_i, z_j) denotes a positive pair and (z_i, z_k) denotes a negative pair.It can be seen that the numerator part of this function encourages the higher similarity of positive examples (the closer the distance in the representation space); the denominator part encourages the lower the vector similarity between any negative examples (the farther the distance). Thus, guided by the InfoNCE loss function, the model can be trained to achieve our desired goal in the optimization process.

SimCLR takes the data in the batch as a negative example. SimCLR's experiments also proved that in the contrastive learning based on negative examples, the more negative examples, the better the effect of the contrastive learning model. We cannot enlarge the batch size infinitely. So a realistic idea is: when we choose negative examples, we are no longer limited to finding negative examples in the batch but the entire unlabeled training data. In the collection, randomly select data of any size as a negative example during model training. This idea is more straightforward to look at, but it is not easy to implement. In 2020, He et al. proposed the typical method of selecting negative examples in the entire training set named MoCo [31].MoCo uses a queue to store and sample negative samples. The queue stores the used feature vectors of previous batches. These stored feature vectors can be used for later optimization steps to enhance the effect of InfoNCE loss. Moco V2 [12] is an improved version of MoCo after absorbing SimCLR's Projector structure and more difficult image augmentation methods. These changes significantly improve the performance of MoCo framework.The augmentation method, encoder structure, projector structure, similarity calculation method, and InfoNCE loss function of MoCo V2 are similar to SimCLR. The most important feature and innovation is the lower branch of the two-branches network. As shown in Fig 2.14, the input of the contrastive learning framework is a pair of augmented examples. The framework also has a two-branch network: the upper branch network will handle the input augmented by transformation t (the primary augmentation), and the lower branch network will handle the input augmented by transformation t' (the secondary augmentation). The upper and lower branches in SimCLR are symmetrical, and the two can

share parameters, while the MoCo V2 lower branch network parameter update uses a momentum update mechanism.

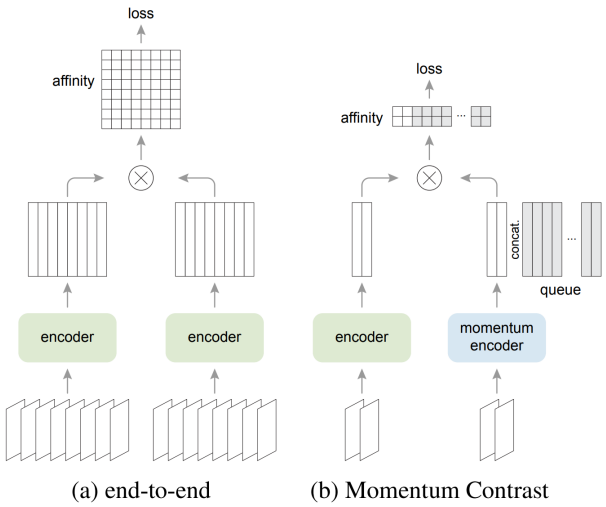


Figure 2.15: A batching perspective of two optimization mechanisms for contrastive learning. Images are encoded into a representation space, in which pairwise affinities are computed. [12]

The lower branch of Moco V2 is also composed of two non-linear mappings of sequence encoder and projector. The structure of these two components is consistent with the corresponding structures of the upper branch, but it has its own set of parameters. The update of this set of model parameters does not use the conventional loss function backpropagation to update the gradient but uses the following Moving Average mechanism to update:

$$\theta_k = m\theta_k + (1 - m)\theta_q \tag{2.7}$$

θ_q is the model parameters of the upper branch, θ_k is the model parameters of the lower branch, and m is the weight adjustment coefficient. When the model training starts, the θ_q is randomly initialized. The upper branch model parameters θ_q are updated through the backpropagation gradient when processing the batches. After θ_q changes, the framework uses the above announcement to update the parameters set θ_k corresponding to the lower branch. Generally, the m will take a large value (0.9 or even 0.99). It means that relative to the upper branch parameters, the lower branch parameters change slowly and steadily, from the random value to the θ_q update bit by bit, and iterate to the optimal value in careful small steps.

MoCo V2 uses a memory bank to keep all processed batches as negative samples. Thus, Moco V2 maintains a large negative example queue. When loss function requires positive

and negative examples, K negative examples will be taken from this queue. The value of K can be adjusted as needed, but unlike SimCLR it has not been limited to the batch size. The momentum update structure of the lower branch has two functions: one is projecting the second augmented image to the corresponding representation space, then providing the representation vector as a positive example to the query vector from the upper branch (extracted from the first augmented image). The second function continuously updates the negative examples in the queue: put the feature vectors from the latest batch into the queue, and the vectors from the oldest batch are out of the queue.

The momentum updating of MoCo V1 and V2 is very similar to the updating method for the target Q-network in deep Q-learning [56]. Experiments show that if parameter m in the momentum update formula is small, the effectiveness of contrastive learning will drop sharply. On the other hand, only when the parameters of the positive and negative examples of the lower branch change slowly and steadily (large m), a better training result can be achieved. The reason can be due to the slowly updated model parameter θ_k, which gives relatively stable and uniform changes to the representation vectors from different batches in the queue, increasing the latent representation space's consistency.

The core idea of SimCLR and MoCo are similar, but compared to SimCLR, MoCo V2 uses data resources more effectively while using fewer hardware resources. The training batch size of SimCLR needs to be 4096 to achieve the best performance in the paper; on the other hand, MoCo V2 cleverly uses the momentum and memory bank mechanisms to make self-supervised contrastive learning have better results with limited hardware resources.

2.4 CNNs for Video Representation Learning

In the era of deep learning, the main workload of video action understanding is how to design a suitable deep network rather than manually designing features. We need to consider two aspects when designing such a deep network: (1) what model can best capture timing and spatial information from existing data. (2) how to reduce the required computation without sacrificing too much accuracy.

2.4.1 Temporal Information Fusion and 2D CNN

Organizing temporal information is a critical point in building a video understanding model based on 2D-CNNs. [40] summarizes the methods for combining temporal information over the video frames (Fig 2.16):

1. Single frame only considers the characteristics of the current frame and only merges

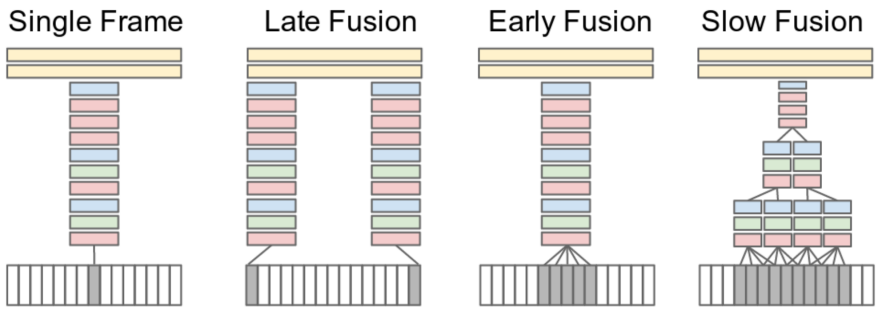


Figure 2.16: Explored approaches for fusing information over temporal dimension through the network. Red, green and blue boxes indicate convolutional, normalization and pooling layers respectively. In the Slow Fusion model, the depicted columns share parameters. [40]

the information of all frames in the final stage.

2. Late fusion uses two feature extraction networks with shared parameters to extract features from two video frames separated by 15 frames. It is also in the final stage that the prediction results of these two frames are combined.

3. Early fusion performs feature fusion on ten consecutive frames in the first layer.

4. Slow fusion's temporal perception field is larger, including inter-frame information fusion in multiple layers. This method is a compromise between early fusion and late fusion.

In the final prediction stage, we can uniformly sample some segments from the video (usually ten segments). Then we can predict the classes of the sampled segments, and the average or vote across all segments will be the final prediction result. However, using 2D CNN for feature extraction at a single frame level then fusing can not model the temporal order of the actions. The temporal order is an important factor to distinguish the actions. For example, sometimes an action can be a reverse of another action (e.g. "open a door" and "close a door"). Considering that the fusion method cannot collect the temporal structure information [10], we must rethink the new temporal information extraction method. Furthermore, we know that these fusion methods are all manually designed ways to fuse features between frames, while deep learning networks only play a role in extracting single frame features. Therefore, this approach may not be effective. We expect to design a deep network that can perform end-to-end learning, utilizing both temporal and spatial information.

We know that optical flow is a manually designed feature of motion information in traditional CV, so the two-stream network [62] combines the RGB image and optical flow of the video as input. In the two-stream network, we also need to sample the entire video sequence to get several segments. Then we calculate the optical flow information from each

segment as the motion information to describe the movement of the action and then sample a frame from this segment to be used as an RGB representative (usually the middle frame, the segment length is usually 10), which characterizes the appearance information of the entire segment. The classification results are obtained in the motion stream and RGB appearance stream, respectively. Finally, the two-stream fusion of the respective classification results is performed in the last layer to obtain the prediction result of the entire segment.

Here we need to pay attention to some technical details. Our optical flow needs to be calculated in two directions. For each frame, the optical flow in the x-direction and the optical flow in the y-direction will be generated. If the fragment length $L = 10$, we will have 20 optical flow channels for each segment, so the input tensor size of our motion stream is $(batch_size, height, width, 20)$, and the input tensor size of appearance stream is $(batch_size, height, width, 3)$. Therefore, motion flow cannot directly use the ImageNet pre-trained weights to initialize because the input channel of the pre-trained model on ImageNet is usually three. However, we can use the convolution kernel (e.g. kernel shape is $(H, W, 3, 64)$) of the first convolution layer in the ImageNet pre-trained model (such as [63]), average the dimension of the RGB input channel (which is the third dimension in the example), then get a size $(H, W, 1, 64)$ tensor. We can use this tensor to initialize the first layer of our motion stream so that although our motion stream has 20 input channels, the parameters of all channels are initialized to the same. After the conversion of the first layer, other subsequent layers can continue to use the pre-trained VGG layer. Therefore, the final output tensor shapes of the motion stream and appearance stream are the same.

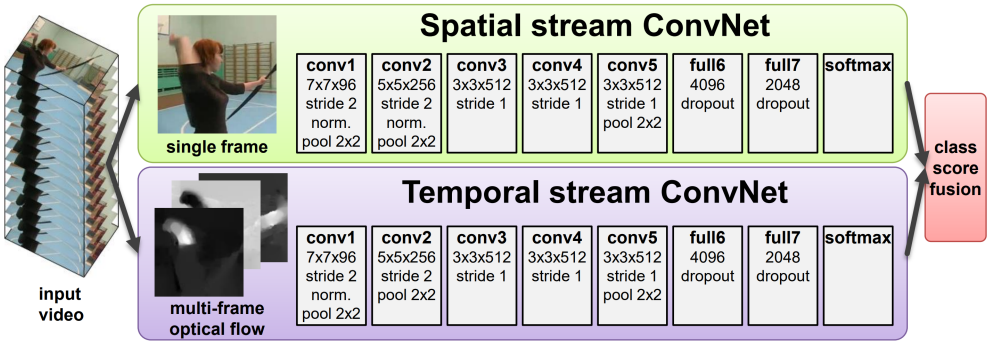


Figure 2.17: Two-stream architecture for video classification. [62]

The problems of the two-stream network include the computational complexity of calculating the optical flow, the content and the label in the sampled segments that may not match, and the long-term dependence on the accuracy of the optical flow information. However, a significant advantage of the dual-stream network is that we can use the ImageNet

pre-trained model as a good parameter initialization because both streams use 2D convolutional networks for feature extraction.

2.4.2 3D-CNNs to Separable Convolution

So far, we have tried to apply a 2D convolution operation to a single video frame for feature extraction and then stack it on the time axis to obtain the final feature containing temporal information. However, if the convolution operation can extract the temporal information while extracting the spatial information, there is no need to stack the temporal features manually. Therefore, C3D [71] uses 3D convolution to obtain video features. In 3D convolution, each convolution kernel has three dimensions, two are used in the spatial domain, and the other is sliding convolution on the time axis. The convolution kernels used in C3D are all the same size $3 \times 3 \times 3$, and it does not involve any hand-designed feature input, so it is completely end-to-end trainable. The authors tried to design the network to be increasingly deeper, finally reaching the SOTA result at that time.

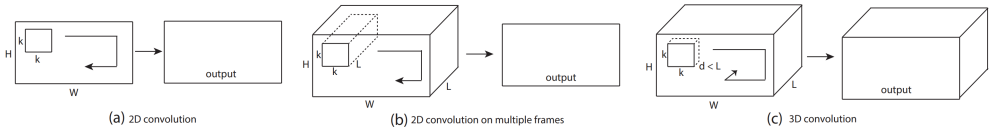


Figure 2.18: 2D and 3D convolution operations. [71]

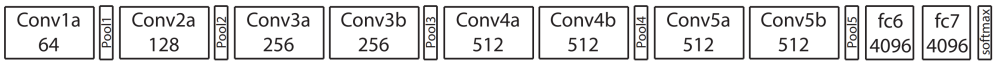
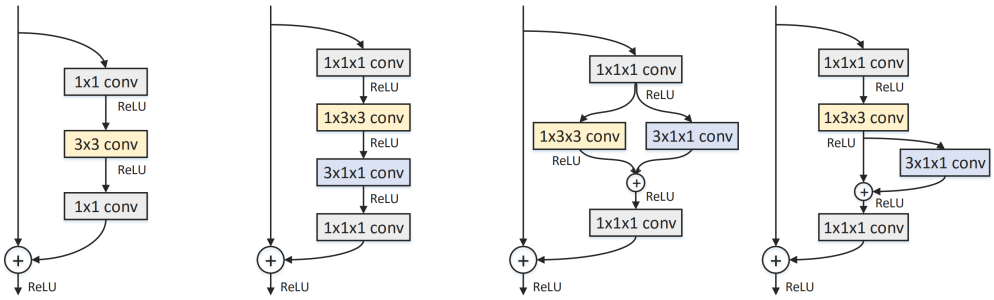


Figure 2.19: C3D architecture. C3D net has 8 convolution, 5 max-pooling, and 2 fully connected layers, followed by a softmax output layer. [71]

However, the biggest problem is that C3D has many parameters, which leads to the need for a large amount of labelled data for training. Moreover, 3D convolution is difficult to pre-train on large-scale image data sets such as ImageNet, which means it has to train from scratch. If the data set is small, then C3D often produces severe overfitting. With the continuous launch of large-scale video action understanding data sets, such as the launch of Kinetics [41], this problem has been alleviated. Carreira and Zisserman [10] further proposed inflating all the 2D convolution filters in an Inception network [68] into 3D convolutions named I3D, and inherit the two streams ideas of [62]: use two networks to take both RGB and optical flow as input, then averaged their predictions. However, I3D is still computationally heavy, making the deployment diffcult and convergence slow. Also, they are more likely to overfit as they have more parameters than their 2D counterparts.

Since 3D CNNs have large computational costs, researchers have been interested in explor-



(a) Residual Unit [7] (b) P3D-A (c) P3D-B (d) P3D-C

Figure 2.20: Bottleneck building blocks of Residual Unit and Pseudo-3D. [60]

ing more efficient models. The technique to save computation is to replace 3D convolution with separable convolutions. Mobile Nets, which have particularly efficient architecture for 2D image based tasks, achieve excellent performance with separable convolutions.The 3D version is similar, except the idea is applied to the temporal dimension instead of the spatial dimention. Pseudo-3D ResNet [60] continues the residual design of ResNet 2.20. This strategy allows the network to be designed deeper, with fewer parameters, but the performance can reach the SOTA results. Not only that, but P3D makes us notice that the number of parameters can be greatly reduced by decomposing 3D convolution into 2D (spatial)+1D (temporal) convolution. This operation also makes it possible to pretrain on the image data set (although this pre-training may not be as intuitive as a 2D network).

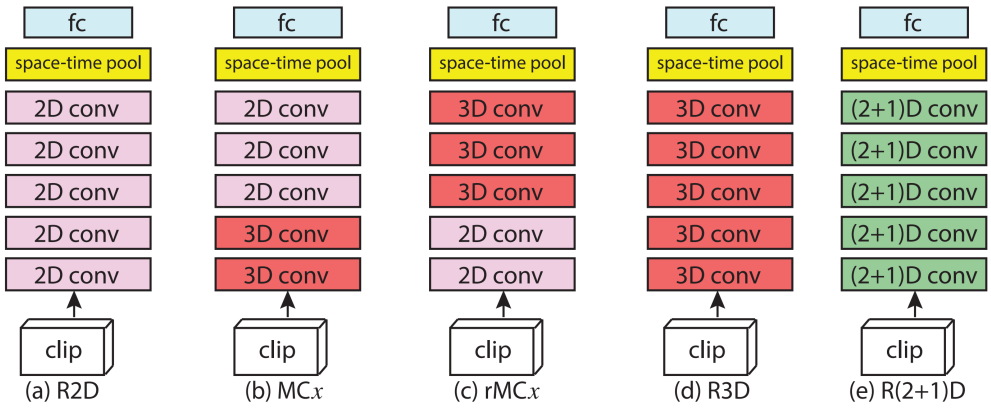


Figure 2.21: Residual network architectures for video classification considered in [72]. (a) R2D are 2D ResNets; (b) MCx are ResNets with mixed convolutions (MC3 is presented in this figure); (c) rMCx use reversed mixed convolutions (rMC3 is shown here); (d) R3D are 3D ResNets; and (e) R(2+1)D are ResNets with (2+1)D convolutions. For interpretability, residual connections are omitted.

In [72], the authors proposed the R(2+1)D network. The author compared a series of different 2D+1D decomposition operations, including a series of 2D+3D operations. The difference from P3D ResNet is that R(2+1)D uses units with the same structure, as shown in 2.4.2, unlike P3D, which has three different residual blocks designs. This design simplifies the design while achieving the SOTA performance.

2.5 Self-supervised Video Representation Learning

Video is a series of semantically related single-frame two-dimensional images superimposed on the time axis, and it is not easy to extract temporal semantic information. Various experiments have proved that existing deep networks do not perform well in extracting temporal semantic features. The main reason is that the amount of labelled video data is insufficient, so self-supervised learning has gradually received attention in video understanding. The insufficient amount of annotated video data does not necessarily reflect the lack of annotations at the video level. There may be many video-level data with action tags, but these videos may not have been cropped, and there is too much unrelated content in the video. It requires intensive manual labour to accurately locate the time when the action occurs in the video. Furthermore, such self-supervised learning can help the networks to learn representations from unlabelled data. Therefore, researchers have recently investigated the transferability of learned video representations from self-supervised tasks to downstream tasks such as action recognition or video retrieval.

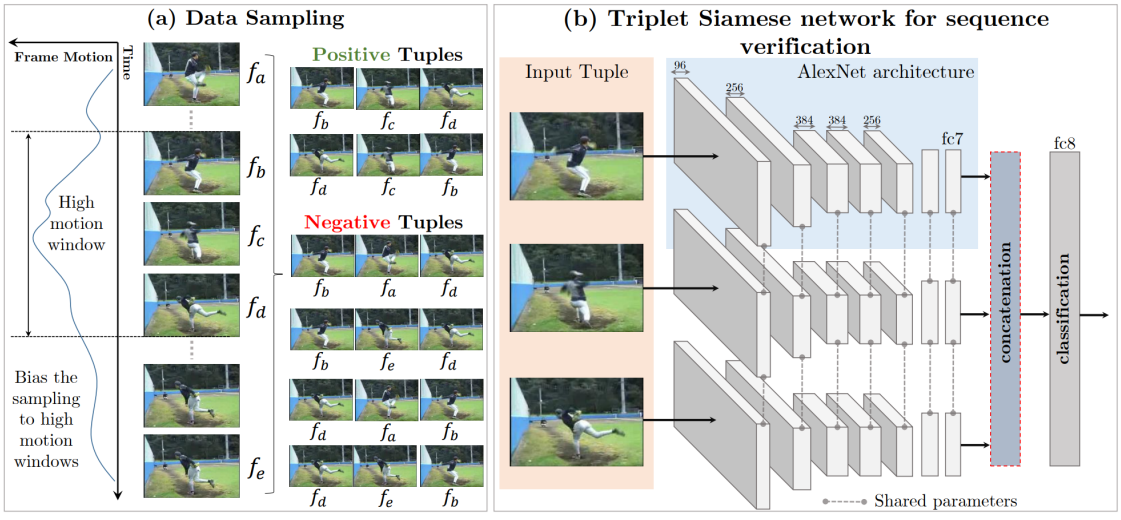


Figure 2.22: An overview of methods for learning characterization by verifying the sequence of video frames. (A) Data sampling process; (b) The model is a three-tuple conjoined network where all input frames share weights. [54]

The video frames will naturally be arranged in chronological order. Researchers have proposed some self-supervised tasks, hoping that a good representation can be learned from the correct frame ordering. One of the ideas is to verify the order of the frames [54]. The pretext task is used to determine whether the frame sequence in the video is arranged in the correct time sequence (time sequence check). The model needs to track and infer the small motion of the object in the entire frame to complete this task.

The training frame is sampled from a window with large motion changes. Each sample gets 5 frames $(f_a, f_b, f_c, f_d, f_e)$, the order of the timestamp is $a < b < c < d < e$. In addition to these five video frames, we also created a positive tuple (f_b, f_c, f_d) and two negative tuples: (f_b, f_a, f_d) and (f_b, f_e, f_d). The pretext task is a binary classification that asks the network to predict whether the temporal order of the input tuple is correct.

Similar work is O3N (Odd-One-Out Network) [20], in which the self-supervision task is also based on the verification of video frame sequences. Given N+1 input video fragments, the video frames in one of the fragments are scrambled and therefore have the wrong frame order, while the remaining N fragments maintain the correct time order. The O3N network learns to predict which video clip is scrambled. In their experiment, the input contained six video segments, and each fragment contained six frames.

The idea of contrastive learning is very effective in the field of self-supervised learning. Models such as MoCo [12] and SimCLR [11] have reached SOTA or even surpassed supervised learning in ImageNet image classification tasks. CoCLR [30] was proposed by Oxford University, which draws on a two-stream network and uses optical flow information to aid self-supervision. As mentioned before, we need both positive and negative samples for each instance to build an input pair for contrastive learning. The authors proposes a self-supervised co-op training method called CoCLR, which stands for "Co-training and Contrastive learning of visual representation." to use other complementary views of the data. The authors chooses the RGB video frame and optical flow as two views. As shown in Fig. 2.23, the positive instances obtained from the stream can be used to "bridge" the gaps between instances of RGB video clips. Conversely, the positive instances obtained from RGB video clips can be linked to optical flow clips of the same action. As a result, the CoCLR algorithm exceeds the performance obtained using InfoNCE for RGB instance-based training only. The paper mainly uses the supplementary information provided by another view to improve the RGB and Flow network representation. In order to make inferences, we can choose to use only the RGB network or the Flow network or use both networks at the same time.

The CoCLR algorithm is carried out in two stages: initialization and rotation. First, use

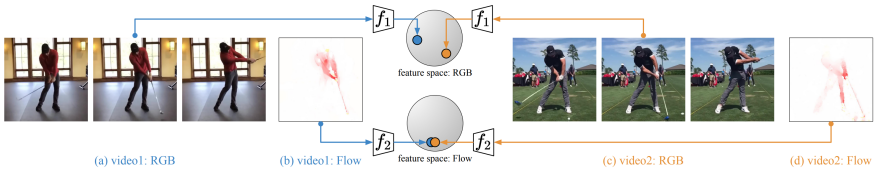


Figure 2.23: Two video clips of a golf-swing action and their corresponding optical flows. In this example, the flow patterns are very similar across different video instances despite significant variations in RGB space. This observation motivates the idea of co-training, which aims to gradually enhance the representation power of both networks, $f_1(\cdot)$ and $f_2(\cdot)$, by mining hard positives from one another. [30]

InfoNCE loss to independently train two models with different views and establish and optimize their respective projection spaces. After training through the InfoNCE loss method, the RGB and flow networks have gained a stronger ability to learn representation than the randomly-initialized network. Then alternate using one network to sample positive pairs for another network, here we take flow to RGB as an example:

- First, obtain the sample of the optical flow.
- Send it to the Flow network to extract features.
- Compare the feature with the features extracted by other flow samples.
- Select the flow top K similar instances in the space (here K is a hyperparameter).
- Then we use these K samples with similar flow features as positive pairs to update the projection space for contrastive learning in the RGB network.

The paper uses two downstream tasks on the datasets UCF101 and HMDB51, action recognition and video retrieval, to evaluate the CoCLR framework. The experiment shows that the latest technology or equivalent performance is superior to other self-supervised methods, and the data usage efficiency is significantly improved; that is, less data is required for self-supervised pre-training.

Recent works like SpeedNet or Pace Prediciton [8, 81] demonstrated that teaching networks the speediness of motion in videos through playing speed classification. Based on simple physics, executing the same task at different speeds causes different motion dynamics. Thus, we can simulate this phenomenon easily by playing videos at different playing rates. The hypothesis is that the networks can only perform playing speed classification tasks when they understand the video content and distinguish the non-trivial motion features (the trivial motion includes motionless, subtle actions etc.). The core idea of playing speed classification is sampling frame clips from original videos using two different frame sampling rates: (1) original FPS and (2) downsampling by skipping one frame. They define the frame clips sampled with actual rate as standard speed clip, and the clips sampled with latter rate as fast speed

clip. During the training, the data loader will randomly sample both frame clips with the constant length as input to the encoder networks, then predicting the input's sampling rate from the encoder's output feature. Traditionally, the learning of motion dynamic was made explicit by computing optical flow between the video frame pairs and identifying the region where several vital attributes (i.e., magnitude and orientation) occurred and changed among the frames. Unfortunately, the cost of computing dense optical flow among the video frames is prohibitive and time-consuming.

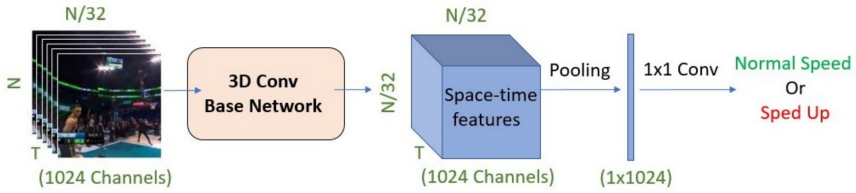


Figure 2.24: SpeedNet architecture, is trained to classify an input video sequence as either normal speed, or sped up. [8]

In SpeedNet's downstream experiments, they transferred the pre-trained weights of standard 3D-CNN (C3D[71], R21D[72] etc.) using Kinetics 400 [41]'s RGB frames then finetuned with the smaller action recognition dataset such as UCF101[65] and HMDB51[45]. They also trained another network that takes per frame average flow magnitude from the Kinetic 400 as input to serve as a flow baseline. The finetuning results of SpeedNets exceed the flow baseline and other prior works and showed that the networks learn reliable spatial-temporal representation.

2.6 Downstream Tasks and Related Datasets

2.6.1 Action Recognition

The research of action recognition is mainly based on the recognition of video data, which mainly includes gesture recognition, sports recognition, and target recognition. Research in all these directions is inseparable from representative video datasets. However, unlike the very mature and commonly used data sets in image recognition, such as MNIST and ImageNet, the data sets in the field of action recognition are relatively limited and usually take a large amount of hard disk space.

UCF sports action dataset [65] is a data set mainly about sports. This data set is also a relatively high-quality data set. The disadvantage is that the number of each type of data is relatively small. In the beginning, there were only 13 regular sports actions. However, as the performance of the video understanding network improved, UCF added other non-sports

actions (such as makeup, dancing, etc.) to the data set and subsequently launched UCF50 (containing 50 categories) and UCF101 (containing 101 categories).

Figure 2.25: UCF101 class examples. [65]

The HMDB dataset [45] has 51 categories, and each category has 100-200 sets of data on average. From the view of data volume and category, it can be seen that it has relatively rich data, but this data set is also built by some movie scenes and daily videos taken by cameras, so the background is relatively complicated, and there are also motion blur and scene switching contents. Therefore, this data set may be more suitable for target recognition and target detection. As an action recognition task, it is more difficult than UCF101.

The Kinetics series of datasets [41] are popular benchmarks in action recognition. As it is a database for downloading YouTube videos, some of the original videos may be removed or modified. Therefore, the data sets used by different users will be slightly different, and it is almost impossible to reproduce the results of others completely. Each video sample of Kinetics data corresponds to an annotation, and the average video length is about 10s. The annotation content includes category name, YouTube video ID (for downloading), video start time (timestamp in YouTube original video), video end time (timestamp in YouTube original video), which dataset it belongs to (training/validation/test set). The Kinetics dataset includes a wide range of categories, with 400 categories (Kinetics400) when it was just launched in 2017. Later, 600 and 700 classes versions (Kinetics600 and Kinetics700) were introduced to test larger neural networks.

2.6.2 Video Retrieval

In video retrieval, we input a query video, and then get related videos from a large video set [1]. The specific operation is to use the network to extract the feature vector of the query video and then compare it with the feature vector extracted from other data in the dataset, select the closest sample, and verify whether it is similar to query video in semantics (or belongs to the same class). Similar to contrast learning, Euclidean distance can also be

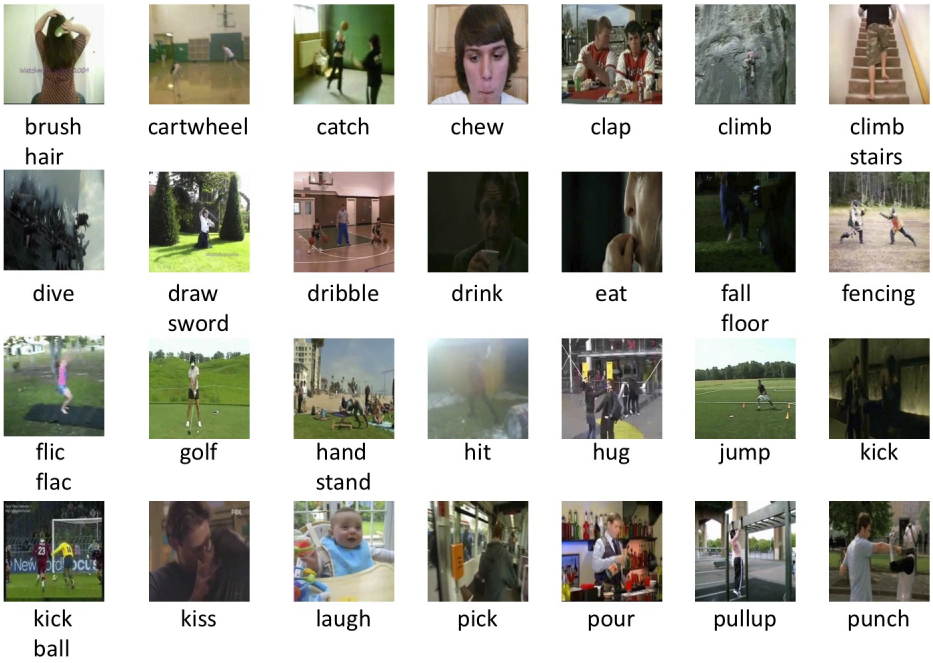


Figure 2.26: Illustration of HMDB51 examples. [45]

used as the calculation of similarity. When this task is used as an evaluation metric for self-supervised learning, it does not require any follow-up training. So the video retrieval can be a good evaluation of the quality of the video representation extracted by the model. The evaluation method is checking whether there are video clips with the same action label in the Top-K samples (usually $K \in [10, 20, 50]$ [8, 81]) closest to the query video in the representation space.

All action recognition datasets introduced above can be used for video retrieval as well. However, there are still some specialized video retrieval datasets (such as MSRVTT [69] or YouCook2 [4]) include the text description of the query. The input of the system can be both video and text or text only. The network has to model both visual data and text data to make the prediction. This thesis will not discuss these video retrieval datasets.

Chapter 3

Speediness Change Classification

3.1 Introduction

When we design pretext tasks for self-supervised learning, the first thing to consider is what kind of tasks the model can complete only after acquiring knowledge about video features. For example, the experimental results of the previous work, Speed Net [8], show that the binary classification of normal speed and double speed video can be completed only after the model has learned the spatiotemporal features well.

But Speed-Net's binary classification task is too simple: it only classifies normal playback speed and double playback speed. We believe that such pretext tasks are not enough to explore the relationship between video playback speed classification and spatiotemporal feature learning. When designing pretext tasks, one rule of thumbs is that if we use a more complex pretext task, the network may learn a better representation space. Therefore, we think it is worth exploring whether Speed-Net's pretext task can improve the efficiency of self-supervised learning when more speed transformations are added. The most obvious way is to add more speediness classes to the system, such as 3x and 4x sped-up video clips, to explore the spatial-temporal features further. Wang et al. [81] achieve better results in downstream tasks through this approach.

On the surface, video clips of different speeds are obtained by data loader using different sampling rates when capturing video frames. But we can also view it as a transformation of a fixed-length video clip in the temporal dimension. Changing the speed is just one of the easiest ways to change. We asked whether it is possible to improve the efficiency of self-supervised learning by adding a new temporal transformation method to the previous classification task. Therefore, we designed two new speed transformation methods in the first experiment: speed jumping up and speed jumping down. When generating pretraining

clips from the original video, we changed the playback speed (i.e. frame sampling rate) at a certain point in the clip. The speed can be either increased (speed jumping up) or decreased (speed jumping down), and we ask the network to classify these two transformations and other constant speeds. The subsequent downstream task evaluation shows that the two newly added transformations are beneficial for the self-supervised learning of video representations.

3.2 Speediness Change Classes

Similar to the previous works, we will not use all video frames in one epoch in order to avoid overfitting. Let $V = \{v_i\}_{i=1}^{N}$ be a video set containing N videos. Each video contains a random number of frames $T = \{t_i\}_{i=1}^{m}, m > 0$. To obtain a single input sample to the network from video set v_i, we sample a frame clip c_i at a random start point t_{start} with constant length **K** from the original video. For a clip c_i sampled from video v_i, our speediness-aware (awareness of the difference among playing speeds) self-supervision strategy aims to learn a spatiotemporal encoder F_{θ_f} parameterized by θ_f that maps c_i to speediness-aware rich feature f_i.

In mechanics, we define the rate of change of the velocity of an object with respect to time as acceleration. However, it is impossible to quantify the video scene's dynamic clearly because the shifting of pixels may not reflect the objects' actual movement speed. Therefore, similar to the SpeedNet, we are not targeting to predict the object's exact acceleration in a video. Typically, there are three usual types of change of pace in our daily life: acceleration (speed jump-up), deceleration (speed jump-down), and constant (no change of pace). We want to design serval frame sampling strategies to mimic these three motion dynamics and randomly apply them to the video clips. Our pretext task will be asking the network to reveal which sampling strategy the speed pattern of the input belongs to. The speediness pattern we used for this approach includes:

Constant Speeds: The first pattern will be sampling video clips with a constant frame rate. To increase the variety, the playing speed rate of the video clips will include both the actual rate and the sped-up rate. As shown in the Fig[3.1], we will sample this class's clips by different downsampling rates p. If $p = 1$, the clips are sampled consecutively from the video (i.e. the playing speed is the same as the original). If $p > 1$, we sample video frames from the original video for every p frames, so the clips' motion will look like "sped-up". As mentioned in Wang's paper[81], it is worth trying out the up-sampled video clips (i.e. slow down the playing speed), but in this case, we need to obtain intermediate frames by interpolation. The interpolation process is costly, and the interpolation error may also affect the inference. Wang slow down the playing rate by inserting repeating frames to the clip. This

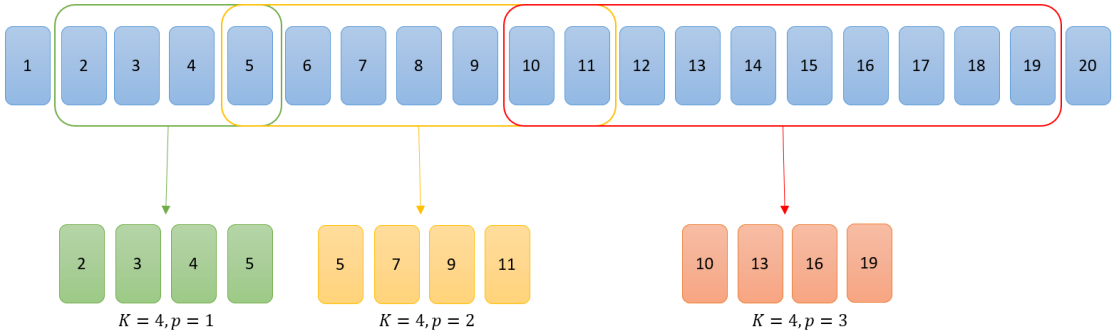


Figure 3.1: Example of sampling constant speed clips with clip length $K = 4$ from the original video frames. The samples start at random start points with random sampling rates.

method can slow down the motion visually but has significant compatibility issues with our proposed approach. Repeating frames can be considered as zero speed, so their interpolation method is sampling the clips with $p = 0$ and $p = 1$ alternatively, instead of sampling with $0 < p < 1$. Since our experiments also aim to identify the clips with variable playing speed, altering sample rates is similar to our speed jump definition. Therefore, we only consider the sampling rates $p \geq 1$ here.

Similar to the Pace Prediction[81] we select the downsampling rate from 1 to 4 for this class. Wang's experiments indicate that more speediness classes can cause prediction task to become too difficult for the network and it fails to learn useful semantic features. Therefore, we follow Wang's maximum constant playing rate as the start. In the later experiments, we also carefully tune the speed-up gap to control the hardness and look for the best settings of the pretraining.

We do not use descriptive words like "normal" or "fast" to describe the video clips sampled with different playing speed rates because the FPS of the video depends on the recording device; it can vary from 8 FPS to 60 FPS or even more. However, we are still asking the network to predict the sampling rate in this approach, which means using the sampling rate as a subclass.

Speed jump-up: Our second sampling pattern tries to simulate the acceleration of the motion. It is difficult to create a smooth, realistically accelerated clip from the video. Therefore we are going to apply different sampling rates when creating a video clip. To sample an accelerating clip, we first select two different sampling rates r_1, r_2 and $r_1 < r_2$ from our constant speed class sampling range, denoted by $p^j = (r_1, r_2), j = r_2 - r_1, 0 < j < 3$. Parameter j is the gap between two sampling rates which also represents the extent of speed

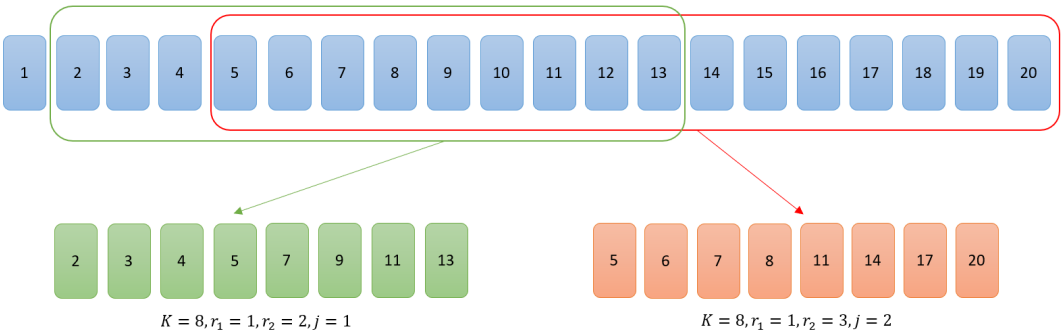


Figure 3.2: Example of sampling speed jump-up clips with clip length $K = 8$.

jump-up. We are using parameter j as the subclass of the speed jump-up clips, and we limit this parameter $0 < j < 3$ to reduce the difficulty of the task. Before the sampling, we choose the parameter j firstly, we use uniformed random sampling from1 to 3 for this parameter in our experiments. Then random sample available (r_1, r_2) pairs based on j (e.g. since we set the range of p between 1 to 4, if $j = 1$ then available r_1 is 1 to 3 and $r_2 = r_1 + 1$, if $j = 2$ then available r_1 is 1 to 2 and $r_2 = r_1 + 2$.) In this approach we always let the change of speed happened at the middle of the clip $c_i[\frac{K}{2}]$, where $c_i[k]$ stands for the image at time position k of clip c_i. During the sampling of the clip c_i with length K(usually a power of 2), we use smaller sampling rate r_1 for the first half of the clip $c_i[0 : \frac{K}{2}]$, and use larger sampling rate $r = 2$ for the second half of the clip $c_i[\frac{K}{2} : K]$ (Fig[3.2]).

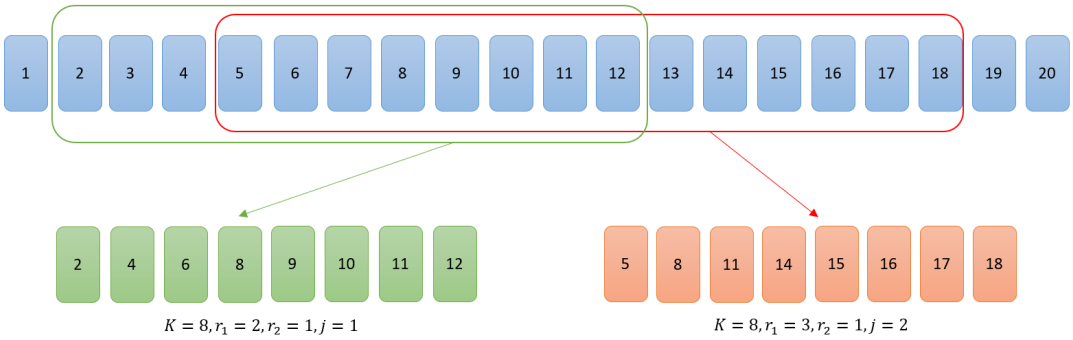


Figure 3.3: Example of sampling speed jump-down clips with clip length $K = 8$.

Speed jump-down: The third pattern is speed jump-down, which is simulating the decrease of the motion dynamic. The method to generate training samples is similar to the speed jump-up pattern except the first sampling rate r_1 is greater than the second one: $p^j = (r_1, r_2), j = r_1 - r_2, 0 < j < 3$. The change of speed happens at the same place as for

the speed jump-up ($c_i[\frac{K}{2}]$). Sampling speed jump-down clips are still using the same range of p which is 1 to 4. As shown in the Fig[3.3], if $j = 1$ then available r_1 is 2 to 4 and $r_2 = r_1 - 1$, if $j = 2$ then available r_1 is 3 to 4 and $r_2 = r_1 - 2$.

We uniformly sample all three transform patterns above during the iteration of the video set. Then we uniformly sample the speediness rate p for the constant speed pattern or the speed jump-up parameter j for the speed jump-up/speed jump-down pattern. We will discuss the detail of the sampling method in the implementation section later. Finally, we are asking the network to predict both the pattern class of the clip and the subclass (i.e. the value of p or j) by using a linear classifier on top of the learned features. The clip size K in Fig[3.4] is only for visualization, we use larger clip size $K = 16$ in our implementation which is a popular clip size in the previous works.

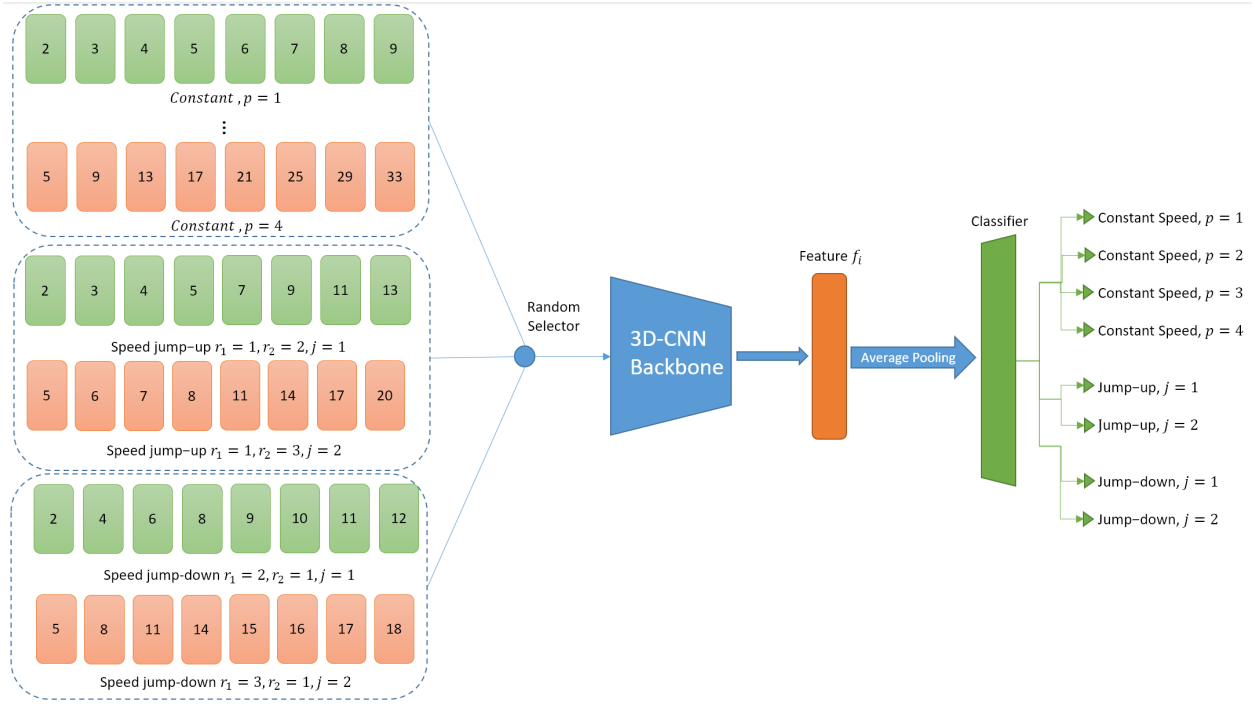


Figure 3.4: A visualization of the classification network in the first approach. For each video sample, we select the transform pattern randomly and generate corresponding video clip with random start frame. The final classifier layer will output a probability distribution over all possible transform patterns.

3.3 Contrastive Learning Module

Similar to Pace-Prediction [81], we use the strategy of contrastive learning. It originate from Noise-Contrastive Estimation loss (NCE loss) [26] and targets to recognize the positive samples from a group of negative samples. If we can define a can further regularize the feature learning. We aim to define a simple binary problem to be solved by the network and store the knowledge through embedding the features from encoder to a latent vector space. The NCE loss can help repel the negative groups from the positive samples in the latent vector space.

An essential issue of contrastive learning is the strategy of designing positive and negative example pairs. We hope the extracted knowledge will help the feature learning process, and use the learned embedding space to regulate the encoder's output. For example, the previous work, Pace Prediction [81], proposed two sampling strategies, same context and same pace. For the same context, if the sample pair comes from the same video (different speed transformation), it is positive; the others are negative. For the same pace strategy, if the sample pair has the same speed transformation (from different videos), it is a positive pair even the clips come from different videos. Our data loader will sample a pair of clips for each input video: one query clip (the primary clip) and a key clip (a sample positive to the query clip). The starting frames of the clips are randomly selected separately.

Based on their experiment results, we chose the same context strategy because the InfoNCE loss 3.2 we used for contrastive learning is more powerful as we have more negative samples to pair with the query clip. The maximum number of possible negative pairs is limited by the speed transformation in the same pace strategy. On the other hand, we can use all clips from other videos to assemble negative pairs if we choose the same context strategy.

Due to the limitations of GPU, we use the MoCo V2 approach [12] to perform contrastive learning. Our data loader will randomly sample a pair of video clips (query and key) from a video in each iteration. We randomly select a speed change class in the previous subsection, then apply the corresponding speed transformation to the query clip and generate the label. We also randomly apply a speed transformation to the key clip, same or different from the query clip, but without generating the labels because only the query clip will perform speed change classification.

Like MoCo V2, our model also consists of the upper and lower branches. Each branch contains a backbone and a projection head. We respectively input query sample and key sample to the upper and lower branches, then combine the projection vectors output by each branch to form a positive pair for the calculation of InfoNCE loss [75]. It should be noted that the key sample only participates in contrastive learning, so the data loader does not

need to generate the label of the key sample. Instead, we only input the feature map from the query backbone to the subsequent speed change classifier, then calculate the cross-entropy loss [2] with the classifier's result and the query sample's label.

Since the original work is established for image inputs, we modify the MoCo V2 framework to adapt the video clip inputs by replacing the original backbones of upper and lower branches to 3D CNNs with 3D pooling layers (e.g. R21D [72], C3D [71]). In addition, we also use 3D convolutional layers to replace the original projection head. We will introduce the details of the network design in later subsection 3.4.

3.4 Network Design and Training Details

3.4.1 Network Overview

Here's a complete neural network visualization for our pretraining. At the beginning of each iteration, the backbone M and projection network M_p in the lower branch will be updated by the upper branch's parameters using the equation (2.7). In each branch, we fix the number of frames in each input clip to 16, then feed the clip to a 3D CNN backbone. We only use the query feature from the upper branch to calculate the cross-entropy loss of the speediness-change classification. At the same time, the query feature and key feature are input to the corresponding projection network. We combine the latent vectors from the two projection networks into a positive sample pair for calculating NCE loss.

3.4.2 Projection Network

Like MoCo-v2, we embed the video representation from the backbone to a latent vector space through a projection network. The projection network extracts the target information (content, speediness, etc.) from the video representation and produces the latent vector to form the positive sample pair. We use the projection network design in the previous work COCLR [30]. Fig 3.6 is a visualization of our projection networks in upper and lower branches. The dimension of the latent vector dim_{proj} is a hyperparameter, and we set it to 128, as same as the previous work.The projection vectors from the upper and lower branches will form a positive sample pair since the two input clips are from the same video.

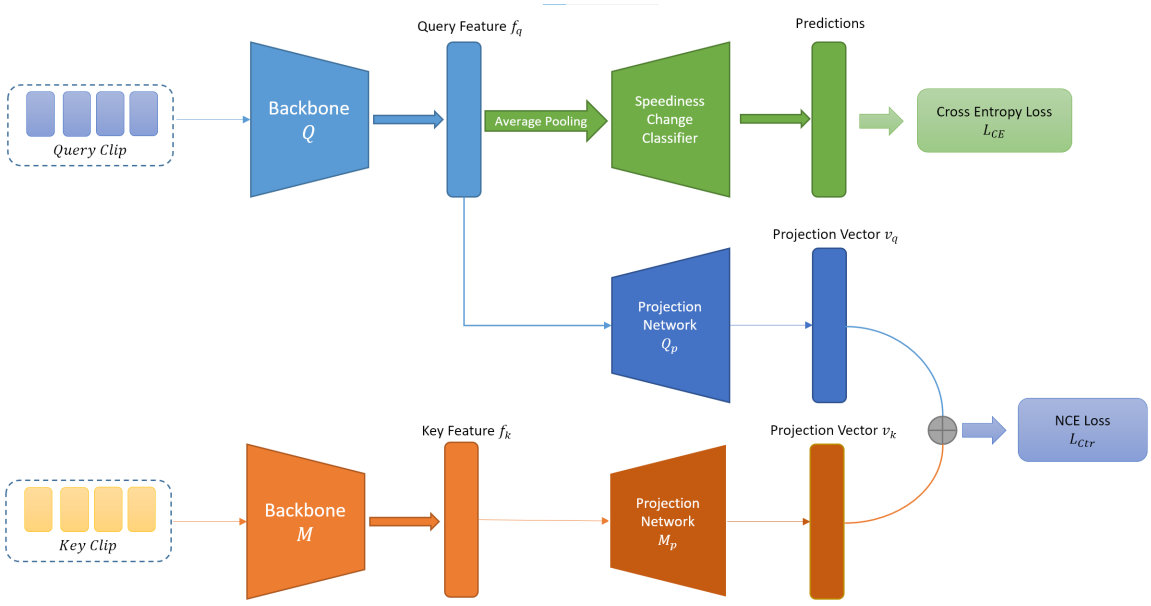


Figure 3.5: A visualization of the classification network in the first approach combining with the MoCo-V2 contrastive learning framework

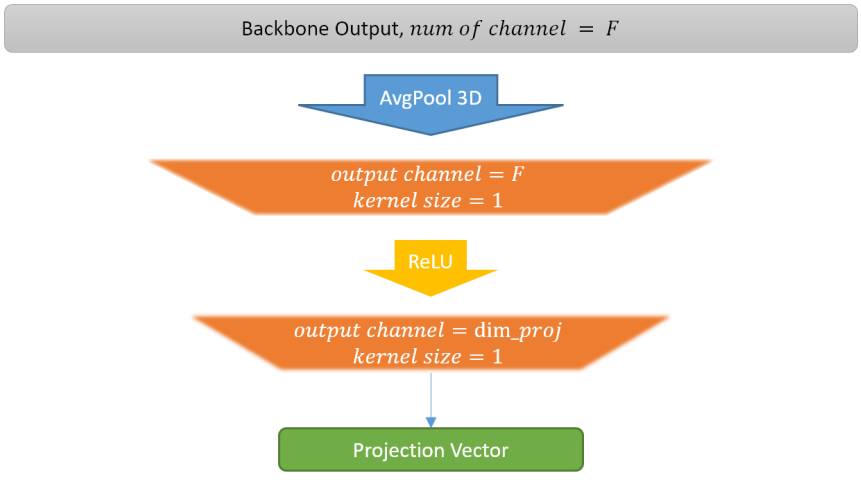


Figure 3.6: A visualization of the projection networks in the upper and lower branches.

3.4.3 Loss Functions

Speediness change classification

For this task, we adopt the same-batch training strategy [8] during training, where each batch contains all transformed clips for each speediness change class from each video sample. The motivation is that, as the batch includes all possible transforms, the model will rely on perceiving global motion consistency of the input clips rather than context information or any other artificial cues to solve the task. We use the cross-entropy loss [2] for the optimization. Assume that there are B samples in one batch, P is the speediness-change predictions from the upper branch and y are the labels, the speediness-change classification loss L_{cls} is,

$$L_{cls}(P, y) = \frac{1}{B}\sum_{i=1}^{B}(-\sum_{i=1}^{c} y_i \times log(P_i)) \tag{3.1}$$

Contrastive loss

We also use the InfoNCE [11, 12] as the loss function for contrastive learning (L_{ctr}). In each iteration, the loss function L_{ctr} will use the positive vector pair from the two-branch network and N negative vector pairs by combining the query vector and all negative samples in the memory bank. The hyperparameter N is the maximum capacity of the memory bank, and we set it to 2048 in all experiments. Identical to the COCLR [30], we use dot product as the vector similarity metric $S(z_i, z_j)$.

$$L_{ctr}(v_q, v_k) = -log\frac{(exp(S(v_q, v_k)/\tau)}{\sum_{i=1}^{K} exp(S(v_q, v_k)/\tau))} \tag{3.2}$$

Joint optimization.

Our self-supervision strategy trains the upper branch backbone Q to generate video representations that are reliable for both pretext tasks and the same-content contrastive learning. We achieve this by jointly optimizing our network with the multi-loss function,

$$L = L_{cls} + \lambda L_{ctr} \tag{3.3}$$

where $\lambda \in [0, 1]$ are the individual weights on the losses. Usually we set the λ between $[0.25, 0.5]$, since the contrastive loss is working similar to a regularization term at here. We first set λ to 0.5 to make two loss terms have similar initial values. Then, we reduce this parameter to 0.25 to get better performance on finetuning for action recognition tasks

through validation. The joint optimization promotes the model to learn speediness-aware rich features that embed the global-local motion patterns and the context information of the video.

3.5 Experiments and Results

3.5.1 Implementation Details and Datasets

To evaluate the efficacy of our proposed method, we choose the following video understanding benchmark datasets that are popularly used for action recognition, i.e. UCF101 [65] and HMDB51 [45]. For UCF101 and HMDB51, we use the training/testing split 1 for a fair comparison to prior works. During the pretraining stage, we use 10% of the training set for validation.

UCF101 action recognition dataset is introduced by [65] in 2012. It includes 13,320 videos, around 27 hours in total. The dataset includes 101 different classes, and the content mainly covers five categories of actions: human-object interaction, human body movements, human-human interaction, playing musical instruments, sports. Videos of each category are divided into 25 groups, and each group contains 4-7 videos. The videos from the same group have similar characteristics, such as background, characters, etc. All videos are from YouTube, 25FPS sample rate, and the average video duration is 7.21 seconds.

The HMDB dataset [45] is one of the most important datasets in the current research field of action recognition, focusing on human behaviour. Brown University released the HMDB51 dataset in 2011. Most of the videos in this dataset come from movies, and some come from online video libraries such as YouTube. The database contains 6849 short videos, 30FPS, divided into 51 categories of human actions, and each class has at least 101 samples.

Although the HMDB dataset has fewer classes, it's a more challenging dataset than UCF101 because the HMDB classification requires the model to learn detailed temporal features of human actions. Compared with HMDB51, UCF101 has one characteristic or shortcomings. The model can classify UCF Sports well using only the spatial features of the scene because many sports activities are carried out in typical scenarios. Such as swimming sports are always in the water, football games are usually played on grass, and skiing on snow. Also, the different samples in the same category are likely to come from the same video group thus have a similar background. Therefore, simply using the UCF database to evaluate the model is not enough to see whether the model has learned good temporal features.

When pretraining with UCF101, we use 25-fps source frames for both pretrained and eval-

uation datasets. The practically-used fps of source videos affect the video clip's time-span for a fixed temporal dimension. The smaller the fps is, the longer time-span the video clip covers. In our experiments, we use UCF101 for pretraining and UCF101, HMDB51 in the action classification task. As the source videos of UCF101 and HMDB51 are 25-fps, all the pretraining and evaluation datasets are prepared with 25-fps.

Common augmentations are applied on input video clips, including color jittering (randomly change the brightness, contrast and saturation, $p = 0.8$), random cropping, random gray-scale (convert the frames to gray-scale images, $p = 0.2$), Gaussian blur (blurring the frames with Gaussian filter, $p = 0.5$) and horizontal flipping ($p = 0.5$). The parameter p is the probability of applying the augmentation to the individual sample.

In model pretraining stage, we use the training set of UCF101 without any annotations. Stochastic gradient descent(SGD) is used for optimization with an initial learning rate of 0.01. For UCF101, the model is pretrained with a batch size of 32 for 400 epochs, and the learning rate is decayed by 0.1 at the 150th, 250th and 350th epoch when the loss plateaus. We let λ_1=1.0, λ_2=0.25 in (4.3). For all self-supervised pretraining and downstream tasks, we set the length of input video clip l_n as 16 with a resolution of 112×112. For contrastive learning, we use the same settings as the MoCo-v2. The maximum capacity of the memory bank $K = 2048$. The temperature $\tau = 0.07$, the momentum updating weight $m = 0.999$ and the dimension of the latent projection space is $dim_{proj} = 128$.

We use several downstream tasks to evaluate the effectiveness of our method.

(1) Action recognition: The model design for the Action recognition tasks has the same backbone (feature encoder) as the pretrained model, so we can directly load the pretrained weights to the action recognition network. Then we append an adaptive spatial-temporal average pooling layer and a fully-connected layer after the pretrained backbone for action recognition. The data processing part will be the same as the pretraining; the only difference is that now we use the cross-entropy loss of action classification to optimize the network. The model is finetuned end-to-end on the training set of UCF101, HMDB51 for 200 epochs with a batch size of 16. We optimize with SGD with an initial learning rate of 0.01, which is decayed by 0.1 at the 80th and 160th epoch when the loss plateaus.

After the finetuning stage, we have to infer the action labels for the videos in the testing set. To facilitate evaluation and comparison with previous works, we follow the inference protocol in [81, 37, 80]. First, we sample ten 16-frames clips uniformly from each testing video in the UCF101 and HMDB51. We only apply constant center cropping on the frames for each clip and no other data augmentations. Secondly, we input the clips to the finetuned model and get classification probabilities. At last, we average the softmax probabilities of

all clips in the video to generate the predicted action label.

(2) Video retrieval: We also follow the previous work's evaluation protocol for the video retrieval task [81, 37, 80]. Although this task does not require a finetune stage, we must infer the 1-D feature vector for each testing video using the pretrained backbone. After inputting the frame clip, we extract the 3-D feature output from the backbone (the encoding part) then apply a 3-D average pooling to obtain the 1-D feature. For each video in the query set or retrieval target set, we uniformly sample ten crops then extract and average the ten features.

Before performing the nearest-neighbour retrieval, we first normalize the features using the training set's statistics. Then, cosine distance is utilized as the similarity metric. Next, query videos are taken from the test split of UCF101 or HMDB51, and all the videos of the train split are considered retrieval targets. Finally, we consider a query video is correctly classified if its k-nearest neighbours contain at least one video from the correct class. We evaluate our method on UCF101 and HMDB51 with K = 1, 5, 10, 20 and 50. Recall at top-K (R@k) is used as the evaluation metric.

3.5.2 Ablation Study

To determine the positive effects of each pretext task, we conduct ablation studies with R(2+1)D [72] network and evaluate with action recognition task on UCF101. In this section, we fix the random seeds and we use the training/testing split 1 in the UCF101 dataset and 10% of the training set is used as a validation set during the pretraining. We re-implement the pace prediction work [81] with the constant speediness classes $[1, 4]$ as a baseline (i.e.the speediness-change classes without the speed jump-up classes and speed jump-down classes). We also re-implement the MoCo-v2 and using only the same context contrastive learning as the pretext task (CL). The primary purpose of this experiment is to serve as a baseline for combining speediness pretext tasks and contrastive learning. At last, we have a fine-tuned action recognition network with random initialized backbone (without pretext pretraining) to observe how pretext task pretraining help the action recognition task.

We have to train our re-implementation since the original author did not give the complete code and model parameters. Still, from the results of finetuning, our re-implement results are close to the original paper or even higher. After that, we first try to combine the previous constant speeds classes and speediness change classes for pretraining. According to the results of finetuning, the classification of action recognition has improved after adding new categories, just like the hypothesis.

However, the help of these four new categories is minimal because the accuracy rate is

Table 3.1: Ablation studies of our first approach on action recognition task - its sensitivity to the new speed jump-up and speed jump-down classes, the individual contribution of each pretext task. "baseline", "SC" and "CL" respectively denotes the pace prediction baseline, our speediness change classification task and contrastive learning.

Method	Pretrain Classes	UCF101 (Accuracy %)
Random Init	–	57.9
baseline	constant speed	75.1
SC	speed jump	72.8
SC	constant speed & speed jump	**75.8**
CL	–	75.6
baseline + CL	constant speed	76.7
SC + CL	constant speed & speed jump	**77.2**

only enhanced by 0.7% under the same environment and parameters. And when we only use the new four speediness changes classes as the classification task of the pretraining, the result of finetuning drops to 72.8%. For the above results, our analysis is that the new speediness change classification is an oversimplified pretext task. From the second row of table 3.1, we can observe that even if the model does not know the constant speediness information, it can still complete the speediness change classification task.

We believe that the reason is that the model mainly focuses on the spatial information before and after the speed change point when completing the task. Although we added a little disturbance (-1/+1) to the speed change position, it can still be regarded as an artificial signal due to the low randomness. Therefore, the model does not need to pay attention to the entire input clip when inferring; only some frames and speed change points are enough to complete the task.

When the classification task combines constant speed classes and speed change classes, the model needs to extract more temporal information from the input clips to reduce loss. However, due to the existence of speediness change points, the task's difficulty drops rapidly in subsequent epochs, which weakens the representation of the feature. Therefore, it doesn't help much for the following action recognition task.

The second part of the table 3.1 shows the experimental results of combining contrastive learning with speediness classification. The results are in line with our expectation: the same context contrastive learning strengthens the feature representation of the model in the spatial dimension. Applying the MoCO-v2 framework can make the model have higher action recognition accuracy after finetuning on the UCF101 database. The improvement on the baseline setting is +1.6, which is consistent with the results on the original paper [81]. For our method, the improvement brought by contrastive learning is relatively low (+1.4). Speediness change classification also requires the model to learn spatial features to

recognize changes in playback speed, which may overlap with the information obtained from contrastive learning.

We initially conjectured that adding contrastive learning would reduce the model's performance on the original pretext task. This is because the model needs to simultaneously minimize the classification task's loss functions and the contrastive loss. However, whether it is the baseline or our method, the accuracy of the original speediness classification has significantly been improved after adding the contrastive learning module. We think the cause is that the model has obtained more information in the spatial dimension of the video from contrastive learning. This extra spatial information can help the model better understand the temporal order and speediness of the video, and the speediness classification is improved by better temporal reasoning.

3.5.3 Evaluation of Self-supervised Representation Learning

To examine the effectiveness of the proposed speediness-aware pretext tasks, we transfer the learned representation to the downstream tasks, including video action recognition, video retrieval and video action localization. For fair and faithful comparison, we pick the current state-of-the art methods that utilize only RGB frames as conjectured that.

Table 3.2: Comparison with SOTA self-supervised approaches on action recognition task with benchmark datasets.

Method	Pretrained	Resolution	Backbone	UCF101	HMDB51
Shuffle[55]	UCF101	227×227	CaffeNet	50.2	18.1
VCOP [84]	UCF101	112×112	R(2+1)D	72.4	30.9
VCP [53]	UCF101	112×112	C3D	68.5	32.5
PacePred [81]	UCF101	112×112	R(2+1)D	75.9	35.9
PRP [86]	UCF101	112×112	R(2+1)D	72.1	35
TempTrans [37]	UCF101	112×112	R(2+1)D	**81.6**	**46.4**
PSPNet [13]	UCF101	112×112	R3D	70.0	33.7
PSPNet [13]	UCF101	112×112	R(2+1)D	74.8	36.8
STS [80]	UCF101	112×112	R(2+1)D	77.8	40.7
Speediness Change	UCF101	112×112	C3D	**71.3**	**36.3**
Speediness Change	UCF101	112×112	R3D	**70.7**	**37**
Speediness Change	UCF101	112×112	R(2+1)D	77.2	41.6

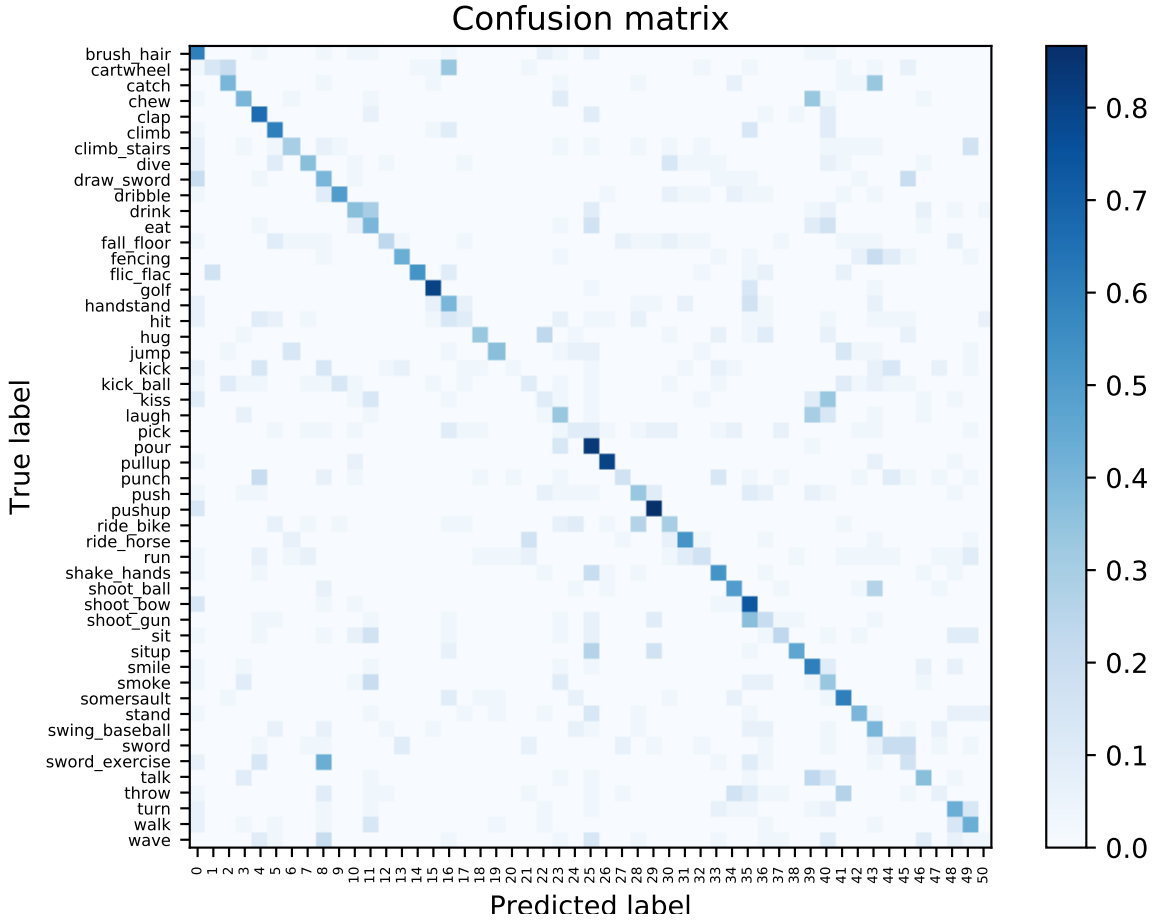


Figure 3.7: Normalized confusion matrix for HMDB51 dataset finetuning

Action recognition.

Table 3.2 shows that our speediness change classification task achieves compatible results over the previous self-supervised methods on both benchmark datasets. We also plot the normalized confusion matrix of HMDB51 dataset classification result for the finetuned R21D model (Figure 3.7). The Y-axis represents the true labels and the X-axis represents the predicted labels (we use class ID on the X-axis). With the same backbone, our models pretrained on UCF101 outperform the original Pace Prediction network [81] and PSPNet [13]. Our method exceeds all previous work when the backbone is the classic C3D [71] and R3D [72] network. But on the R(2+1)D network [72], our speediness-change classification is far surpassed by the latest work: the temporal transformation classification method [37]. Temporal Transformation classification adds other special video frame arrangements such as random, loop and temporal warping based on different playing speeds. And use two classifiers to determine the video playback speed and frame motion type. The pretext task design in this paper inspired our following approach: speediness-change classification and localization (Chapter 4).

Video retrieval.

Table 3.3 presents the results of performing video retrieval task on UCF101. We use the models pretrained on the training split of UCF101. Identical to the previous works ([81, 86, 37, 84, 53] etc.), we uniformly sample ten 16-frames clips from the testing videos as query clips. The retrievals are computed on the videos from the training split for each query clip. The top-k nearest neighbours are queried by computing the cosine distances between two feature vectors. Unlike contrastive learning, the feature vectors here are not coming from the projection network. Instead, we apply 3D average pooling on the backbone network's outputs to get a 1D feature vector. A query is correctly classified if the query class is included in the top-k neighbour. We report the average accuracy for each k value here. Although our video retrieval results underperform the Pace Prediction, they are compatible with other previous works for all backbones.

Table 3.3: Comparison with SOTA self-supervised approaches on video retrieval Recall with UCF101 and HMDB51 dataset.

Method (Pretrained on UCF101)	UCF101/HMDB51				
	R@1	R@5	R@10	R@20	R@50
Backbone: AlexNet					
Jigsaw [58]	19.7/–	28.5/–	33.5/–	40.0/–	49.4/–
OPN [48]	19.9/–	28.7/–	34.0/–	40.6/–	51.6/–
Backbone: C3D					
VCOP [84]	12.5/7.4	29.0/22.6	39.0/34.4	50.6/48.5	66.9/70.1
VCP [53]	17.3/7.8	31.5/23.8	42.0/35.5	52.6/49.3	67.7/71.6
PacePred [81]	**31.9/12.5**	**49.7/32.2**	**59.2/45.4**	**68.9/61.0**	**80.2/80.7**
PRP [86]	23.2/10.5	38.1/27.2	46.0/40.4	55.7/56.2	68.4/75.9
Speediness Change [ours]	26.2/11.2	45.9/30.8	53.6/44.4	62.9/57.8	3.2/77.1
Backbone: R3D					
VCOP [84]	14.1/7.6	30.3/22.9	40.4/34.4	51.1/48.8	66.5/68.9
VCP [53]	18.6/7.6	33.6/24.4	42.5/36.6	53.5/53.6	68.1/76.4
PacePred [81]	23.8/9.6	38.1/**26.9**	46.4/**41.1**	53.5/56.1	69.8/76.5
PRP [86]	22.8/8.2	38.5/25.8	46.7/38.5	55.2/53.3	69.1/75.9
TempTrans [37]	**26.1**/–	**48.5**/–	**59.1**/–	**69.6**/–	**82.8**/–
MemDPC [29]	20.2/7.7	40.4/25.7	52.4/40.6	64.7/**57.7**	–/–
PSPNet [13]	24.6/**10.3**	41.9/26.6	51.3/38.8	62.7/54.6	67.9/**76.8**
Speediness Change [ours]	23.3/11.5	41.6/28.3	52.2/40.6	60.6/54.7	73.3/74.4
Backbone: R(2+1)D					
VCOP [84]	10.7/5.7	25.9/19.5	35.4/30.7	47.3/45.8	63.9/67.0
VCP [53]	19.9/6.7	33.7/21.3	42.0/32.7	50.5/49.2	64.4/73.3
PacePred [81]	**25.6/12.9**	**42.7/31.6**	**51.3/43.2**	**61.3/58.0**	**74.0/77.1**
PRP [86]	20.3/8.2	34.0/25.3	41.9/36.2	51.7/51.0	64.2/73.0
Speediness Change [ours]	22.5/10.1	37.3/26.2	48.1/36.7	59.0/50.7	72.5/69.4

Chapter 4

Multi-task: Speed Change Classification and Localization

4.1 Introduction

The previous chapter introduced speediness change classification, a new pretext task for self-supervised learning of video representation. To evaluate the performance of the new pretext task, we use the pretrained backbone for the two downstream tasks of action recognition and video retrieval. The evaluation results show that our model is compatible with the original Pace Prediction baseline and previous work.

However, we found from the ablation study that the gap between our new method and the baseline is not evident. After pre-training with our newly added speediness change classes, the finetuning result of the backbone has only increased by 0.5%. We believe that the first pretext task we designed did not effectively improve the pre-training. We review our method in Chapter 3 and find two main drawbacks: (1) The playing speed changing point is always next to the center frame, which can be an artificial signal or trivial feature for learning; (2) grouping the speed change clips by speed change $j = r_1 - r_2$ makes the definition of each speed change class ambiguous.

Therefore, we have improved the original speediness-change classification task. At first, we adjust the method of generating the speediness-change clip: the speed transition may occur anywhere in the middle part (between 1/3 and 2/3) of the clip. And we require the model to make a more fine-grained speed classification to increase the difficulty. Secondly, we propose a new pretext task: speediness-change localization. We add a new linear classifier to the original network in parallel. After defining the possible range of speed transition, we ask the model to determine between which two frames the speed transition occurs. Fi-

nally, we repeat the downstream task experiment and ablation study for new pretext tasks. The experimental results are better than the results of our first method and prove that the modified speediness-change classification and speediness-change localization are helpful to increase the pre-training efficiency of self-supervised learning.

4.2 Speed Change Recognition

Our ablation study in chapter 3 shows that the new variational speediness transformation will improve video representation learning at the pretext task stage, compared to the previous work Pace Prediction [81]. However, the improvement is minor. Therefore, we revisited the speediness-change classification in Chapter 3 and found inadequacies.

The first drawback of our original setting is that the transition points of the speeds are not sufficiently random: the speediness-change clip sampling always changes the pace around the center frame of the video clip. This construction method may be an artificial signal because the model can focus only on the central frame of the clips. Once a noticeable change of frame difference is detected near the center frame, classifying speediness-change classes can be easy. The system may rely on this trivial feature as more video clips are processed.

Secondly, in the previous chapter, we group all clips with the same playing speed change $j = r_1 - r_2$ into one category, such as $(1, 2), (2, 3), (3, 4)$ are classified to speed jump-up($j = 1$). But this grouping can make the speediness-change classes very hard to learn. The varied appearance across different initial speeds make our speed change classes have a significant variance, and our model struggles to capture the critical features of each speed change class.

Therefore, we decompose our previous speed change classification; each possible initial speed r_1 and second speed r_2 pair will be treated as individual classes. For example, if the possible playing speeds are still in $[1, 2, 3, 4]$, the previous jump-up ($j = r_1 - r_2 = 1$) class will be decomposed to three jump-up classes $(1, 2), (2, 3), (3, 4)$, and the previous jump-up ($j = 2$) class will be decomposed to two jump-up classes $(1, 3), (2, 4)$. The constant speed classes will remain the same, so the total number of classes is $4^2 = 16$ classes. Our ablation study shows that we can get better downstream task results if we reduce the possible playing speeds to $r_1, r_2 \in 1, 2, 3$, so our final number of speediness-change classes is 9, listed in (4.1).

Also, our new clip generator will uniformly randomly sample the speed change point among the middle $\frac{1}{2}$ part of the clip. If the length of the clip is not divisible by 3, we round $\frac{K}{3}$ and $\frac{2K}{3}$ down to an integer. For example, if the input length is 16, the speediness-change point is randomly decided between the fifth and tenth frames. We do not want a range larger than $\frac{1}{3}$ because the model may not have enough frames to infer the speed of r_1 or r_2. Since our speediness-change point is now varied across different input clips, we add a new pretext task

named speediness-change localization and stack it with speediness-change classification. We introduce the detail of speediness-change localization in section 4.3.

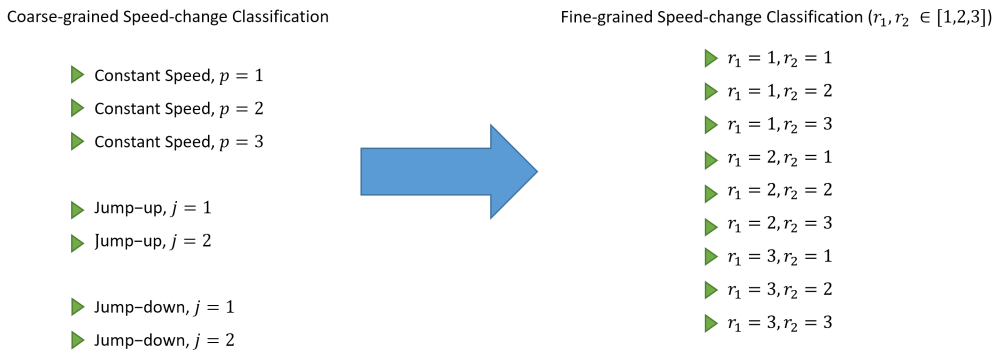


Figure 4.1: Comparing the coarse-grained speediness-change classes and fine-grained speediness-change classes.

4.3 Localizing the Speed Change

As explained in section 4.2, we will randomly choose the speed change position $c_i[m]$ in the clip for our second approach. The random positioning augments the input clips for the speediness-change classification, avoids the overfitting and removes the possible artificial signal. We cannot set the change point too early or late; otherwise, the network may not acquire enough information from the first or second part of the clip to infer the speed.

After pretraining the backbones with our new speediness-change classification, we get improved finetuning results. Therefore, we develop a new idea: use the model to locate the speediness-change position as a pretext task parallel to the speediness-change classification. The localization task can find the dividing frame of the first and second speeds, which is helpful for the classification task. Furthermore, the classification task will learn about different playing speeds, which helps the localization task notice the temporal difference of the first and second parts in the input clip.

We hypothesize that these tasks can complement each other by finding the speediness-change location and identifying the two playing speeds. The parallel pretext tasks will help the pretraining process converge quickly and allow the model to learn more fine-grained (more motion change details) features. Besides, the transformations in past works are mainly applied at the whole clip or video level and provide coarse-grained labels for supervision. In contrast, the proposed method operates at a local temporal point (i.e. where the speed change occurs). This distinction encourages our approach to exploit frame-wise temporal coherence and capture more fine-grained motion changes.

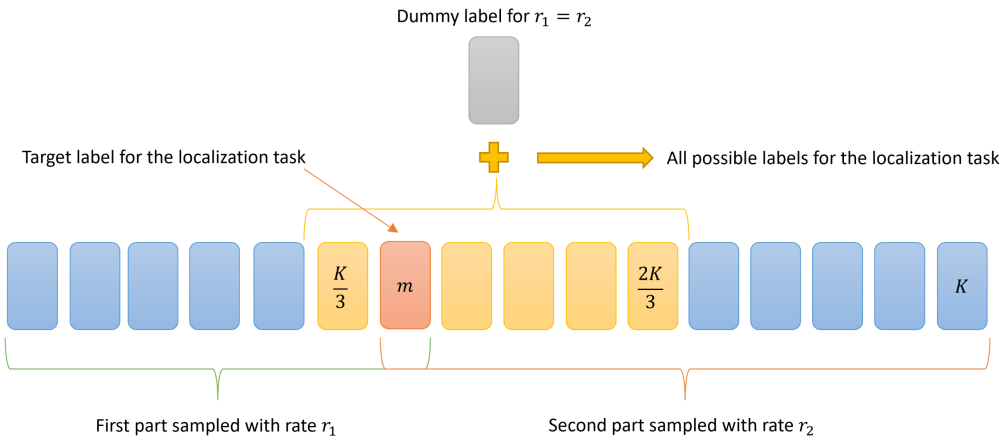


Figure 4.2: Visualization of speediness-change localization labels.

In our implementation, we sample the speediness-change position m from the range $[\frac{K}{3} : \frac{2K}{3}]$ which is also the label for the localization task. We use discrete classification to localize the speed change point instead of regression since it is hard to use real numbers to describe frame positions.

We add one dummy class label for the constant speed clips (when $r_1 = r_2$, the localization label will always be the dummy label), so the total number of the labels is $\frac{K}{3} + 1$. The label for the acceleration localization is even more fine-grained than the labels in the classification task. Thus, it drives the network to perceive more subtle and fine-grained motion changes and complements the representation learning of the shared backbone.

4.4 Network Design and Training Details

4.4.1 Overview

Here's a visualization of our new pretraining process, Figure 4.3. The overall structure remains the same as in the previous chapter, and we still apply the MoCo-V2 framework with the projection networks. The main difference is that the network will use the output feature from the backbone Q to perform two parallel pretext tasks. First, the original speediness-change classifier is replaced by the fine-grained one, as described in section 4.2. We add an extra fully-connected layer parallel to the speediness-change classifier to perform the speediness-change localization. The speediness-change localization is interpreted as a

classification problem, and we are trying to find out which frame is the start of the second-speed (r_2) part of the input frame.

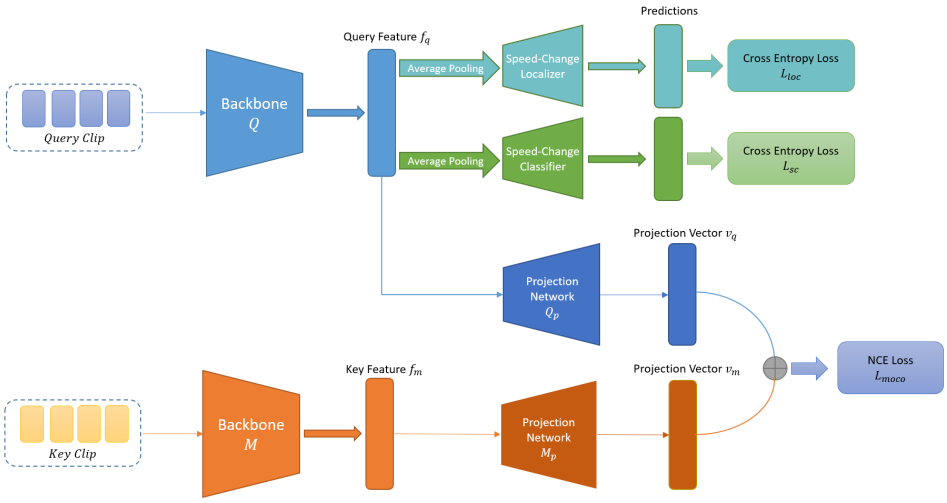


Figure 4.3: Visualization of our new multi-tasks pretraining.

4.4.2 Multi-task Classifiers

We list the pretraining classes we used in our final model in Figure 4.4. The speediness-change classes are similar to the new classes described in section 4.2. Since we use 16-frame input clips, our speediness-change localization has seven possible outputs. The range of speediness-change point, or the start frame of the second part, is fixed between the fifth and tenth frames. We need one more dummy class for the constant speed clips, in which it is impossible to find the speediness-change point.

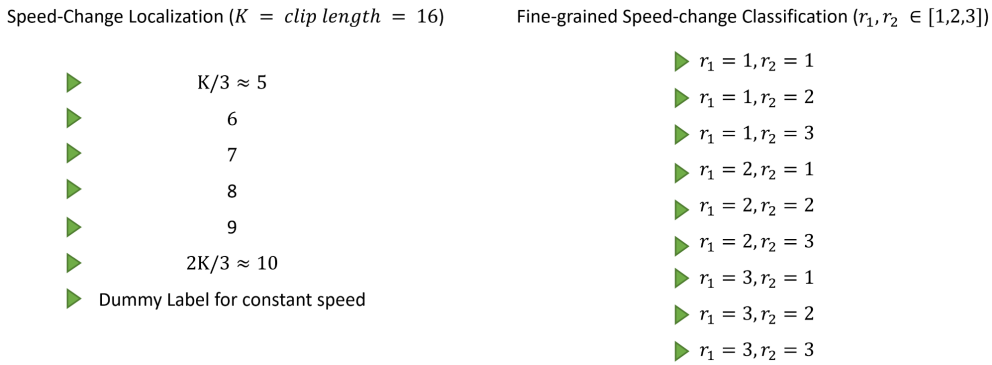


Figure 4.4: The training classes for our speediness-change classification and localization pretext tasks.

4.4.3 Loss Functions

Fine-grained speediness change classification

For this task, we reuse the same loss function for our coarse-grained speediness-change classification in Chapter 3. The only difference is the number of classes. We still use the cross-entropy loss [2] for the optimization. If the range of playing speeds $r_1, r_2 \in [1, 2, 3]$, number of classes c_{sc} will be 9. Assume that there are B samples in one batch, P is the predictions from the upper branch and y are the labels, the fine-grained speediness-change classification loss L_{sc} is,

$$L_{sc}(P, y) = \frac{1}{B}\sum_{i=1}^{B}(-\sum_{i=1}^{c_{sc}} y_i \times log(P_i)) \tag{4.1}$$

Speediness change localization

We don't want the output of speediness-change localization to be floating point numbers; otherwise, it will be hard to generate corresponding input clips and labels. So instead, we convert the localization problem to finding the start frame of the second speed. Therefore, speediness-change localization is a classification problem, and we can still optimize it with cross-entropy loss. If the length of input clip $K = 16$, number of classes c_{loc} will be 7, as visualized in 4.4. Assume that there are B samples in one batch, P is the predictions from the upper branch and y are the labels, the fine-grained speediness-change localization loss L_{loc} is,

$$L_{loc}(P, y) = \frac{1}{B}\sum_{i=1}^{B}(-\sum_{i=1}^{c_{loc}} y_i \times log(P_i)) \tag{4.2}$$

Joint optimization.

Similar to the final loss function in Chapter 3, We jointly optimize our network with the multi-loss function for completing the speediness-change classification and localization tasks and the contrastive learning,

$$L = L_{loc} + L_{sc} + \lambda L_{ctr} \tag{4.3}$$

where $\lambda \in [0, 1]$ are the individual weights on the losses for contrastive learning. However, we get our best validation results when we set the $\lambda = 0.5$ for the new pretraining approach.

We test unbalanced weights for speediness-change localization and classification tasks, but the finetuning results do not outperform the balanced setting. Therefore, the loss weights of these two pretext tasks are the same (1.0).

4.5 Experiments and Results

4.5.1 Implementation Details and Datasets

We continue to use the implementation and hyperparameters we used in Chapter 3. The pretraining dataset is still UCF101. First, we pretrain the backbones using the same optimization (SGD) and scheduler settings for 400 epochs. Then we finetune the pretrained backbones for the action recognition task using UCF101 and HMDB51. We also test the output features from the pretrained backbones to evaluate the quality of representation learning. The details of the two downstream tasks: action recognition and video retrieval, are described in section 3.5.1. All settings and the downstream tasks (action recognition and video retrieval) evaluation protocol are the same as section 3.5.1, except the loss weights for contrastive learning $\lambda = 0.5$ now.

However, we use a different clip sampling strategy in our new approach. The clip generator will uniformly sample the clips with temporal transformations for each speediness-change class in the previous method. Since we have two pretext tasks now, we must modify the sampling method. We can uniformly sample either the speediness-change location within the defined range (middle part frames and dummy class) first, then uniformly sample the available speediness-change class (r_1, r_2), or do it in reverse. The former sampling strategy can make the backbones pretrained with more speed-change clips and less constant speed clips because the generator will produce the constant speed clips only when the first sampler gives a dummy label. The later strategy will define the speediness-change first. If $r_1 = r_2$, the speediness-change location sampler will only produce a dummy label. We test both sampling strategies and pick the former one in the end, based on the downstream task evaluation.

4.5.2 Ablation Study

Firstly, we run a series of pretraining experiments to find out the best settings for our pretraining process and use the finetuning accuracy of UCF101 action recognition as an evaluation metric. We deploy both pretext tasks (speediness-change classification and localization) in these experiments to compare different settings. The results are listed in the table 4.1.
As section 4.5.1 mentioned, we want to decide which pretext task label will be defined first, localization or classification. Row 1 in the table 4.1 shows that the finetuning accuracy drops

Table 4.1: We try to find the best settings for our pretext tasks and better sampling strategy through finetuning the pretrained model on UCF101 action recognition dataset. We use both pretext tasks in these experiments (speediness-change classification and localization).

Primary Sampling	Speediness	Pretext Tasks	UCF101 (Accuracy %)
classes	1,2,3	fine-grained classification + localization	78.2
locations	1,2	fine-grained classification + localization	79.3
locations	1,2,3	fine-grained classification + localization	**79.7**
locations	1,2,3,4	fine-grained classification + localization	78.9

down when defining the speediness-change class first. We believe the drop is caused by insufficient pretraining on the speediness-change localization. Defining the speediness-change class first is more likely to generate a constant speed clip. Class-primary uniform sampling has a 1/3 chance to produce a constant speed clip, and location-primary uniform sampling only has a 1/7 probability of making a constant speed clip. Unlike the clips with speediness-change, the constant speed clips with dummy speediness-change localization labels do not help improve the localization task. More speediness-change clips will lead the model to learn the motion details through the localization task. This experiment is also evidence to prove how our new speediness-change localization task and speediness-change clips improve the effectiveness of video representation pretraining.

We tried different playing speed pools (the possible values for r_1 and r_2) for fine-grained speediness-change classification. When we reduce the playing speeds pool to $[1, 2]$, the finetuning accuracy drops to 79.3%. The fine-grained speediness-change classification with a small playing speeds pool might become a too-easy task for pretraining because the number of classes reduces to four ($[(1, 1), (2, 2), (1, 2), (2, 1)]$). At the same time, the probability of generating speediness-change clips is reduced to 0.5, so our model has less effective pretraining samples for the localization task.

However, the finetuning accuracy decreases after increasing the pool size to $[1, 2, 3, 4]$. Since the number of classes will exponentially increase or decrease with different playing speed pools, increasing the pool size to four can enhance the complexity and hardness of the classification task. It will also increase the hardness of the localization task because each possible position has more variations (combination of different speeds). For a harder pretext task, we might need more pretraining epochs or a more complex backbone [6].

Table 4.2 includes and compares the ablation study with our different speediness-change tasks. We borrow the baseline experiment results from Chapter 3 (random initialize, Pace Prediction re-implement and contrastive learning). We also add the finetuning results of the models pretrained by our previous speediness-change classification in Chapter 3 but re-

Table 4.2: Ablation studies of our new approach on action recognition task - demonstrate the effectiveness of our new fine-grained speediness-change classification and localization. "baseline", "SC" and "CL" respectively denotes the pace prediction baseline, our speediness change classification task and contrastive learning. We rename the classification pretext task in Chapter 3 to coarse-grained classification, in order to distinguish it from the fine-grained classification we introduced in this Chapter.

Method	Pretrain Classes	UCF101 (Accuracy %)
Random Init	–	57.9
baseline	constant speed	75.1
CL	–	75.6
SC	coarse-grained classification	75.8
SC	fine-grained classification	77.5
SC	fine-grained classification + localization	**79.7**
baseline + CL	constant speed	76.7
SC + CL	coarse-grained classification	77.2
SC + CL	fine-grained classification + localization	**81.6**

name it to coarse-grained classification to avoid confusion. The evaluation shows that both new pretext tasks improve the pretraining process, with or without the contrastive learning module. Our final version model achieves 81.6% action recognition accuracy on the UCF101 dataset.

4.5.3 Evaluation of Self-supervised Representation Learning

Same as Chapter 3, we apply new pretext tasks to different 3D-CNN backbones (R(2+1)D [72], R3D [72], C3D [71]) and evaluate the pretrained model with two downstream tasks, action recognition finetuning and video retrieval. After finetuning, we compared the model's performance on downstream tasks with our method in Chapter 3 and previous work that only use RGB video clips as input.

Action Recognition

Both the R3D and C3D models pretrained using our new pretext tasks outperform the previous work in the table 4.3 on the UCF101 classification task. The performance of the R(2+1)D backbone on UCF101 is also comparable to SOTA, Temporal Transformation [37].Furthermore, all backbones far outperform all previous work in the list on another commonly used action recognition database, HMDB51.

Again, we plot the HMDB51 classification task's normalized confusion matrix for the fine-tuned R21D model. We compare the new matrix to the similar plot in Chapter 3 (Fig. 3.7, the colours on the diagonal pixels are deeper in the new plot (Fig. 4.5b), which means the

Table 4.3: Comparison with SOTA self-supervised approaches on action recognition task with benchmark datasets.

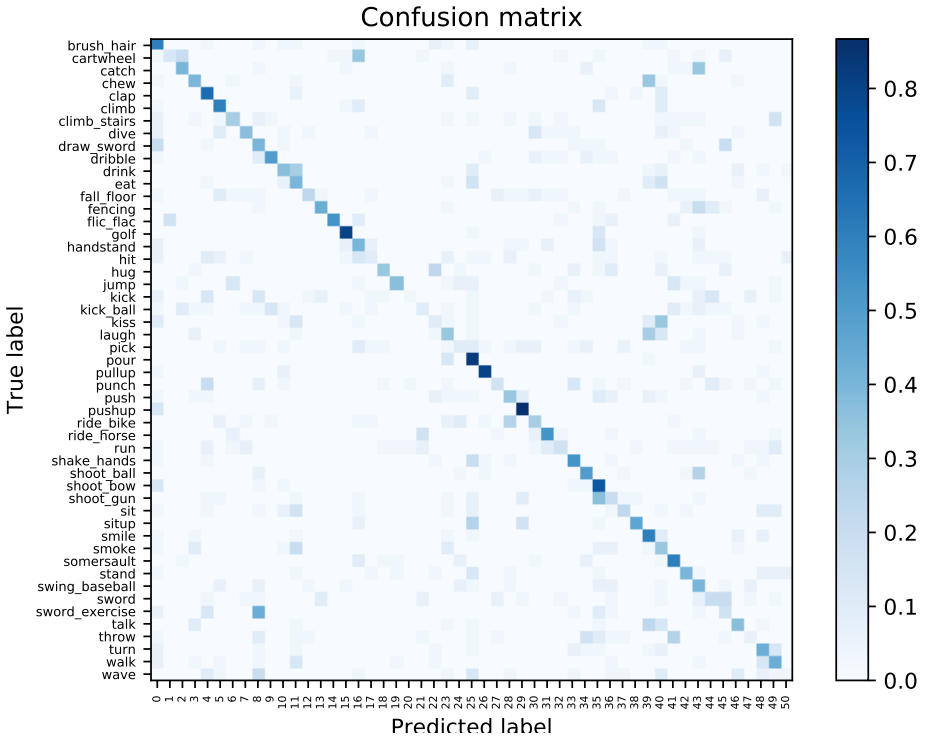
Method	Pretrained	Resolution	Backbone	UCF101	HMDB51
Shuffle[55]	UCF101	227×227	CaffeNet	50.2	18.1
VCOP [84]	UCF101	112×112	R(2+1)D	72.4	30.9
VCP [53]	UCF101	112×112	C3D	68.5	32.5
PacePred [81]	UCF101	112×112	R(2+1)D	75.9	35.9
PRP [86]	UCF101	112×112	R(2+1)D	72.1	35
TempTrans [37]	UCF101	112×112	R(2+1)D	**81.6**	**46.4**
PSPNet [13]	UCF101	112×112	R3D	70.0	33.7
PSPNet [13]	UCF101	112×112	R(2+1)D	74.8	36.8
STS [80]	UCF101	112×112	R(2+1)D	77.8	40.7
Ours (Chapter 3)	UCF101	112×112	C3D	71.3	36.3
Ours (Chapter 3)	UCF101	112×112	R3D	70.7	37
Ours (Chapter 3)	UCF101	112×112	R(2+1)D	77.2	41.6
Ours (Chapter 4)	UCF101	112×112	C3D	**78.3**	**43.8**
Ours (Chapter 4)	UCF101	112×112	R3D	**79.0**	**57.1**
Ours (Chapter 4)	UCF101	112×112	R(2+1)D	**81.6**	**51.5**

new pretraining method improves the classification accuracy of the finetuning. Based on the confusion matrix plots, the R21D model pretrained by the new speediness multi-tasks delivers better feature representation for some hard human interactions such as kissing, kicking a ball etc.

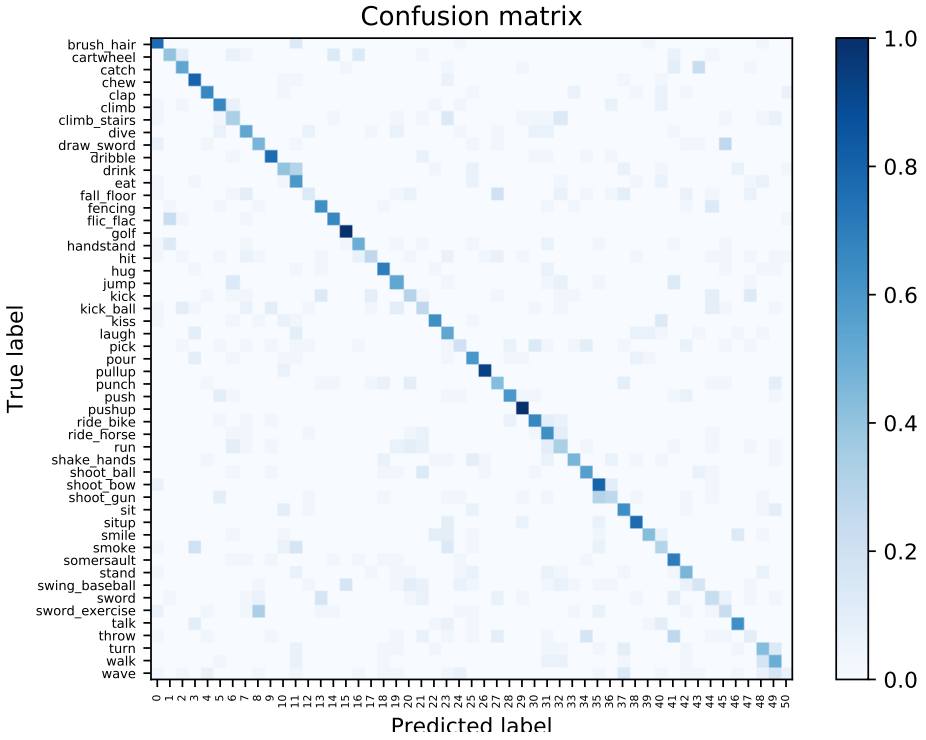
This proves that the video representation learned by our new method has good transferability between different databases. The original paper of R3D and R(2+1)D [72] suggests that the R3D network is more likely to produce training error because the spatial-temporal decomposition implemented by R(2+1)D eases the optimization. The new pretext tasks we introduced in this chapter reduce the performance gap between R3D and R(2+1)D, which indicates that optimizing the speediness-change classification and localization at the same time will not worsen convergence but helps the model's learning process.

Video Retrieval

Same as Chapter 3, we use the pretrained backbones to generate the video representation vectors for the UCF101 and HMDB51 datasets (4.4). We compare the vectors through cosine similarity to find the nearest neighbours to perform video retrieval task. The performance of the three backbones in video retrieval is significantly improved compared to the method in Chapter 3. These results show that our new approach can better learn the temporal and spatial features of the video in the pretraining stage. Our new video retrieval results are compatible with the previous works. All top-K recall scores on the HMDB51 dataset



(a) HMDB51 dataset finetuning result from Chapter 3



(b) HMDB51 dataset finetuning result using new multi-task pretraining

Figure 4.5: Normalized confusion matrices comparison for HMDB51 dataset finetuning

Table 4.4: Comparison with SOTA self-supervised approaches on video retrieval Recall with UCF101 and HMDB51 dataset.

Method (Pretrained on UCF101)	UCF101/HMDB51				
	R@1	R@5	R@10	R@20	R@50
Backbone: AlexNet					
Jigsaw [58]	19.7/–	28.5/–	33.5/–	40.0/–	49.4/–
OPN [48]	19.9/–	28.7/–	34.0/–	40.6/–	51.6/–
Backbone: C3D					
VCOP [84]	12.5/7.4	29.0/22.6	39.0/34.4	50.6/48.5	66.9/70.1
VCP [53]	17.3/7.8	31.5/23.8	42.0/35.5	52.6/49.3	67.7/71.6
PacePred [81]	**31.9**/12.5	49.7/32.2	**59.2**/45.4	**68.9**/61.0	80.2/**80.7**
PRP [86]	23.2/10.5	38.1/27.2	46.0/40.4	55.7/56.2	68.4/75.9
Ours	30.9/**15.1**	**50.3/33.7**	59.1/**46.6**	67.7/**62.3**	**80.3**/79.7
Backbone: R3D					
VCOP [84]	14.1/7.6	30.3/22.9	40.4/34.4	51.1/48.8	66.5/68.9
VCP [53]	18.6/7.6	33.6/24.4	42.5/36.6	53.5/53.6	68.1/76.4
PacePred [81]	23.8/9.6	38.1/26.9	46.4/41.1	53.5/56.1	69.8/76.5
PRP [86]	22.8/8.2	38.5/25.8	46.7/38.5	55.2/53.3	69.1/75.9
TempTrans [37]	26.1/–	**48.5**/–	**59.1**/–	**69.6**/–	**82.8**/–
MemDPC [29]	20.2/7.7	40.4/25.7	52.4/40.6	64.7/57.7	–/–
PSPNet [13]	24.6/10.3	41.9/26.6	51.3/38.8	62.7/54.6	67.9/76.8
Ours (Chapter 3)	23.3/11.5	41.6/28.3	52.2/40.6	60.6/54.7	73.3/74.4
Ours (Chapter 4)	**30.0/15.8**	46.9/**35.2**	47.1/**48.5**	65.8/**61.7**	76.9/79.2
Backbone: R(2+1)D					
VCOP [84]	10.7/5.7	25.9/19.5	35.4/30.7	47.3/45.8	63.9/67.0
VCP [53]	19.9/6.7	33.7/21.3	42.0/32.7	50.5/49.2	64.4/73.3
PacePred [81]	**25.6/12.9**	**42.7/31.6**	51.3/43.2	**61.3/58.0**	**74.0/77.1**
PRP [86]	20.3/8.2	34.0/25.3	41.9/36.2	51.7/51.0	64.2/73.0
Ours (Chapter 3)	22.5/10.1	37.3/26.2	48.1/36.7	59.0/50.7	72.5/69.4
Ours (Chapter 4)	24.7/12.6	42.0/31.4	**51.7/43.7**	61.2/57.6	73.4/76.1

from our UCF101-pretrained R3D backbone outperform all previous works with the same backbone, proving the transferability of learned video feature representation.

4.5.4 Visualization Analysis

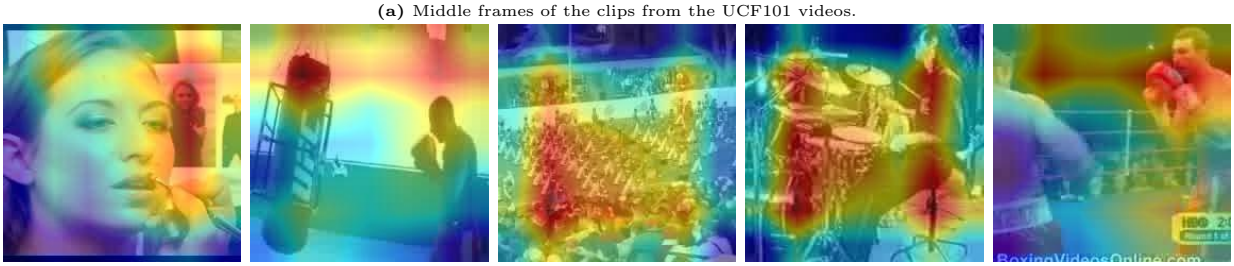
To gain a better understanding of what spatial-temporal features are learned during pre-training, we visualize the salient regions that contribute the most to the proposed tasks, with the class activation mapping (CAM) [91] technique. Similar to [82], we add some small modifications to let the original CAM method adapt the video clip inputs. Specifically, we sample some video clips from the UCF101 videos and extract the feature maps before average pooling layers in the projection networks of the pretext tasks (the tasks in Chapters 3 and

4). The backbone architectures for both pretext tasks are R21D network [72] and without supervised finetuning. We average the feature map along the channel dimension and then resize the compressed features along spatial dimensions to the size of original video frames and overlay on them.

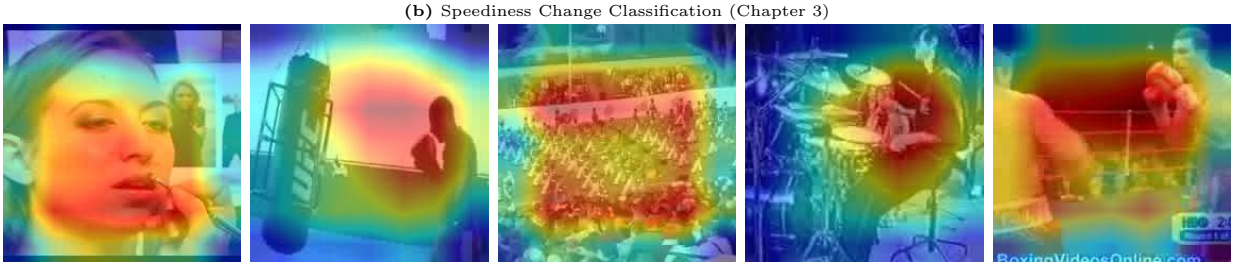
In Fig. 4.6, the middle frame of each clip (Fig. 4.6a) is used to visualize heatmaps. The examples of both pretexts tasks suggest a strong correlation between highly activated regions and the dominant moving objects in the scene. The first and the second examples are bad cases for the previous pretext task (Fig. 4.6b). The activated regions in these two examples deviate from the main moving object to the background. In Fig. 4.6c, we can notice that these bad cases are improved by the model pretrained with the new multi-tasks pretext task, which explained the improvement in accuracy of UCF101 classification.

We also notice that Chapter 3's approach concentrates on the fine-grained motions in the background. Such as the fourth example of Fig. 4.6b, the highly-activated region covering the drums indicates that the model was attracted by the trivial movement of the drum kit (trembling). In comparison, the multi-tasks approach has more significant salient regions over the dominant moving object in the video. To complete the speediness localization, the model must pay more attention to the action of the main subject in the video instead of the background.

(a) Middle frames of the clips from the UCF101 videos.



(b) Speediness Change Classification (Chapter 3)



(c) Speed Change Classification and Localization (Chapter 4)

Figure 4.6: Visualization of salient regions for each speediness-aware pretext task

Chapter 5

Conclusion and Future Work

We propose a series of new pretraining methods for self-supervised video representation learning in this thesis. Our speediness-change classification and localization pretext tasks help 3D-CNN backbones to learn critical video features. These speediness-aware pretext tasks, particularly the speediness-change localization task, are easy-to-implement and effective individually in video representation learning. For performance gains, they can also complement prior self-supervision arts for image representation learning such as MoCo-V2. The pretraining backbone captures fine-grained motion and context features by deploying these tasks together, leading to competitive or SOTA performances on multiple downstream video understanding tasks.

Our extensive experiments and ablation studies in Chapter 4 demonstrate how our fine-grained speediness-change classification and localization improve the effectiveness of pretraining to get better action recognition accuracies than baseline or coarse-grained speediness-change classification. In addition, all pretrained 3D-CNN backbones show good dataset transferability in the downstream tasks. However, due to the limitation of computing resources, we do not pretrain the backbones with large-scale video datasets like Kinetics or Activity-Net. Therefore, we plan to examine whether our speediness pretext tasks can help models learn more comprehensive video features through pretraining the backbones with large-scale datasets or videos extracted from the web in the future. We are also interested in replacing the 3D-CNN backbones with vision transformer networks (e.g. [6]). The vision transformer is a novel architecture for image and video representation learning, inspired by the classic transformer design [76] in the natural language processing area. This new architecture and its variations demonstrate exemplary performance on Image-net or Knietcis datasets in recent papers [6, 18]. We wish to experiment whether using the vision transformer as a backbone could bring improvements to our self-supervised learning framework

Bibliography

[1] Content-based image retrieval. https://en.wikipedia.org/wiki/Content-based_image_retrieval. Accessed: 2021-11-10.

[2] Cross entropy. https://en.wikipedia.org/wiki/Cross_entropy. Accessed: 2021-11-10.

[3] Linear classifiers. https://sites.google.com/site/machinelearningnotebook2/classification/binary-classification/linear-classifiers. Accessed: 2021-11-30.

[4] Youcook2 dataset. http://youcook2.eecs.umich.edu/leaderboard. Accessed: 2021-11-09.

[5] Yubo An and Shenghui Zhao. A video summarization method using temporal interest detection and key frame prediction, 2021.

[6] Anurag Arnab, Mostafa Dehghani, Georg Heigold, Chen Sun, Mario Lučić, and Cordelia Schmid. Vivit: A video vision transformer, 2021.

[7] Mohanad Babiker, Othman O. Khalifa, Kyaw Kyaw Htike, Aisha Hassan, and Muhamed Zaharadeen. Automated daily human activity recognition for video surveillance using neural network. In *2017 IEEE 4th International Conference on Smart Instrumentation, Measurement and Application (ICSIMA)*, pages 1–5, 2017.

[8] Sagie Benaim, Ariel Ephrat, Oran Lang, Inbar Mosseri, William T Freeman, Michael Rubinstein, Michal Irani, and Tali Dekel. Speednet: Learning the speediness in videos. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 9922–9931, 2020.

[9] Michael Bendersky, Lluis Pueyo, Jeremiah Harmsen, Vanja Josifovski, and Dima Lepikhin. Up next: Retrieval methods for large scale related video suggestion. *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 08 2014.

[10] Joao Carreira and Andrew Zisserman. Quo vadis, action recognition? a new model and the kinetics dataset. In *proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 6299–6308, 2017.

[11] Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey E. Hinton. A simple framework for contrastive learning of visual representations. *CoRR*, abs/2002.05709, 2020.

[12] Xinlei Chen, Haoqi Fan, Ross B. Girshick, and Kaiming He. Improved baselines with momentum contrastive learning. *CoRR*, abs/2003.04297, 2020.

[13] Hyeon Cho, Taehoon Kim, Hyung Jin Chang, and Wonjun Hwang. Self-supervised visual learning by variable playback speeds prediction of a video. *IEEE Access*, 2021.

[14] Jin Choi, Yong-il Cho, Taewoo Han, and Hyun Yang. A view-based real-time human action recognition system as an interface for human computer interaction. pages 112–120, 09 2007.

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

[16] Ali Diba, Mohsen Fayyaz, Vivek Sharma, Amir Hossein Karami, Mohammad Mahdi Arzani, Rahman Yousefzadeh, and Luc Van Gool. Temporal 3d convnets: New architecture and transfer learning for video classification, 2017.

[17] Carl Doersch, Abhinav Gupta, and Alexei A. Efros. Unsupervised visual representation learning by context prediction. *CoRR*, abs/1505.05192, 2015.

[18] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. *CoRR*, abs/2010.11929, 2020.

[19] Alexey Dosovitskiy, Jost Tobias Springenberg, Martin A. Riedmiller, and Thomas Brox. Discriminative unsupervised feature learning with convolutional neural networks. *CoRR*, abs/1406.6909, 2014.

[20] Basura Fernando, Hakan Bilen, Efstratios Gavves, and Stephen Gould. Self-supervised video representation learning with odd-one-out networks. *CoRR*, abs/1611.06646, 2016.

[21] Victor Foo, Siang Fook, Pham Thang, That Htwe, Qiang Qiu, Aung aung Phyo wai, Maniyeri Jayachandran, Jit Biswas, and Philip Yap. Automated recognition of complex agitation behavior of dementia patients using video camera. pages 68 – 73, 07 2007.

[22] Spyros Gidaris, Praveer Singh, and Nikos Komodakis. Unsupervised representation learning by predicting image rotations. *CoRR*, abs/1803.07728, 2018.

[23] Sorin Mihai Grigorescu, Bogdan Trasnea, Tiberiu T. Cocias, and Gigel Macesanu. A survey of deep learning techniques for autonomous driving. *CoRR*, abs/1910.07738, 2019.

[24] Chunhui Gu, Chen Sun, David A. Ross, Carl Vondrick, Caroline Pantofaru, Yeqing Li, Sudheendra Vijayanarasimhan, George Toderici, Susanna Ricco, Rahul Sukthankar, Cordelia Schmid, and Jitendra Malik. Ava: A video dataset of spatio-temporally localized atomic visual actions, 2018.

[25] Malik Ali Gul, Muhammad Haroon Yousaf, Shah Nawaz, Zaka Ur Rehman, and Hyung-Won Kim. Patient monitoring by abnormal human activity recognition based on cnn architecture. *Electronics*, 9(12), 2020.

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

[27] Song Han, Xingyu Liu, Huizi Mao, Jing Pu, Ardavan Pedram, Mark A. Horowitz, and William J. Dally. EIE: efficient inference engine on compressed deep neural network. *CoRR*, abs/1602.01528, 2016.

[28] Song Han, Huizi Mao, and William J. Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding, 2016.

[29] Tengda Han, Weidi Xie, and Andrew Zisserman. Memory-augmented dense predictive coding for video representation learning. In *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part III 16*, pages 312–329. Springer, 2020.

[30] Tengda Han, Weidi Xie, and Andrew Zisserman. Self-supervised co-training for video representation learning. *arXiv preprint arXiv:2010.09709*, 2020.

[31] Kaiming He, Haoqi Fan, Yuxin Wu, Saining Xie, and Ross Girshick. Momentum contrast for unsupervised visual representation learning, 2020.

[32] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015.

[33] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016.

[34] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network, 2015.

[35] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *CoRR*, abs/1704.04861, 2017.

[36] Satoshi Iizuka, Edgar Simo-Serra, and Hiroshi Ishikawa. Globally and locally consistent image completion. *ACM Transactions on Graphics*, 36:1–14, 07 2017.

[37] Simon Jenni, Givi Meishvili, and Paolo Favaro. Video representation learning by recognizing temporal transformations. In *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XXVIII 16*, pages 425–442. Springer, 2020.

[38] Shuiwang Ji, Wei Xu, Ming Yang, and Kai Yu. 3d convolutional neural networks for human action recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(1):221–231, 2013.

[39] Andrej Karpathy, George Toderici, Sanketh Shetty, Thomas Leung, Rahul Sukthankar, and Li Fei-Fei. Large-scale video classification with convolutional neural networks. In *CVPR*, 2014.

[40] Andrej Karpathy, George Toderici, Sanketh Shetty, Thomas Leung, Rahul Sukthankar, and Li Fei-Fei. Large-scale video classification with convolutional neural networks. In *2014 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1725–1732, 2014.

[41] Will Kay, Joao Carreira, Karen Simonyan, Brian Zhang, Chloe Hillier, Sudheendra Vijayanarasimhan, Fabio Viola, Tim Green, Trevor Back, Paul Natsev, Mustafa Suleyman, and Andrew Zisserman. The kinetics human action video dataset, 2017.

[42] Prannay Khosla, Piotr Teterwak, Chen Wang, Aaron Sarna, Yonglong Tian, Phillip Isola, Aaron Maschinot, Ce Liu, and Dilip Krishnan. Supervised contrastive learning. *CoRR*, abs/2004.11362, 2020.

[43] Alexander Kläser, Marcin Marszalek, and Cordelia Schmid. A spatio-temporal descriptor based on 3d-gradients. 09 2008.

[44] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'12, page 1097–1105, Red Hook, NY, USA, 2012. Curran Associates Inc.

[45] H. Kuehne, H. Jhuang, E. Garrote, T. Poggio, and T. Serre. Hmdb: A large video database for human motion recognition. In *2011 International Conference on Computer Vision*, pages 2556–2563, 2011.

[46] I. Laptev and Tony Lindeberg. Space-time interest points. volume 64, pages 432–439 vol.1, 11 2003.

[47] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[48] Hsin-Ying Lee, Jia-Bin Huang, Maneesh Singh, and Ming-Hsuan Yang. Unsupervised representation learning by sorting sequences. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 667–676, 2017.

[49] Yuanze Lin, Xun Guo, and Yan Lu. Self-supervised video representation learning with meta-contrastive network, 2021.

[50] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3431–3440, 2015.

[51] David G. Lowe. Distinctive image features from scale-invariant keypoints. *Int. J. Comput. Vision*, 60(2):91–110, November 2004.

[52] Dezhao Luo, Chang Liu, Yu Zhou, Dongbao Yang, Can Ma, Qixiang Ye, and Weiping Wang. Video cloze procedure for self-supervised spatio-temporal learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 11701–11708, 2020.

[53] Dezhao Luo, Chang Liu, Yu Zhou, Dongbao Yang, Can Ma, Qixiang Ye, and Weiping Wang. Video cloze procedure for self-supervised spatio-temporal learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 11701–11708, 2020.

[54] Ishan Misra, C Lawrence Zitnick, and Martial Hebert. Shuffle and learn: unsupervised learning using temporal order verification. In *European Conference on Computer Vision*, pages 527–544. Springer, 2016.

[55] Ishan Misra, C. Lawrence Zitnick, and Martial Hebert. Unsupervised learning using sequential verification for action recognition. *CoRR*, abs/1603.08561, 2016.

[56] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin A. Riedmiller. Playing atari with deep reinforcement learning. *CoRR*, abs/1312.5602, 2013.

[57] John Nickolls, Ian Buck, Michael Garland, and Kevin Skadron. Scalable parallel programming with cuda: Is cuda the parallel programming model that application developers have been waiting for? *Queue*, 6(2):40–53, March 2008.

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

[59] Deepak Pathak, Philipp Krähenbühl, Jeff Donahue, Trevor Darrell, and Alexei A. Efros. Context encoders: Feature learning by inpainting. *CoRR*, abs/1604.07379, 2016.

[60] Zhaofan Qiu, Ting Yao, and Tao Mei. Learning spatio-temporal representation with pseudo-3d residual networks. *CoRR*, abs/1711.10305, 2017.

[61] Holger R Roth, Dong Yang, Ziyue Xu, Xiaosong Wang, and Daguang Xu. Going to extremes: Weakly supervised medical image segmentation, 2020.

[62] Karen Simonyan and Andrew Zisserman. Two-stream convolutional networks for action recognition in videos. *CoRR*, abs/1406.2199, 2014.

[63] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[64] Hassan Soliman, Ahmed Saleh, and Eman Fathy. Face recognition in mobile devices. *International Journal of Computer Applications*, 73:13–20, 07 2013.

[65] Khurram Soomro, Amir Roshan Zamir, and Mubarak Shah. UCF101: A dataset of 101 human actions classes from videos in the wild. *CoRR*, abs/1212.0402, 2012.

[66] Yu Sun, Eric Tzeng, Trevor Darrell, and Alexei A. Efros. Unsupervised domain adaptation through self-supervision. *CoRR*, abs/1909.11825, 2019.

[67] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015.

[68] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott E. Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. *CoRR*, abs/1409.4842, 2014.

[69] Ganchao Tan, Daqing Liu, Meng Wang, and Zheng-Jun Zha. Learning to discretely compose reasoning module networks for video captioning. *CoRR*, abs/2007.09049, 2020.

[70] Martin Thoma. A survey of semantic segmentation. *CoRR*, abs/1602.06541, 2016.

[71] Du Tran, Lubomir Bourdev, Rob Fergus, Lorenzo Torresani, and Manohar Paluri. Learning spatiotemporal features with 3d convolutional networks. In *Proceedings of the IEEE international conference on computer vision*, pages 4489–4497, 2015.

[72] Du Tran, Heng Wang, Lorenzo Torresani, Jamie Ray, Yann LeCun, and Manohar Paluri. A closer look at spatiotemporal convolutions for action recognition. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pages 6450–6459, 2018.

[73] Marco Alexander Treiber. *An Introduction to Object Recognition: Selected Algorithms for a Wide Variety of Applications*. Springer Publishing Company, Incorporated, 1st edition, 2010.

[74] Jen-Kai Tsai, Chen-Chien Hsu, Wei-Yen Wang, and Shao-Kang Huang. Deep learning-based real-time multiple-person action recognition system. *Sensors*, 20(17), 2020.

[75] Aäron van den Oord, Yazhe Li, and Oriol Vinyals. Representation learning with contrastive predictive coding. *CoRR*, abs/1807.03748, 2018.

[76] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *CoRR*, abs/1706.03762, 2017.

[77] Olga Veksler. *Regularized Loss for Weakly Supervised Single Class Semantic Segmentation*, pages 348–365. 10 2020.

[78] Pascal Vincent, Hugo Larochelle, Y. Bengio, and Pierre-Antoine Manzagol. Extracting and composing robust features with denoising autoencoders. pages 1096–1103, 01 2008.

[79] Heng Wang and Cordelia Schmid. Action recognition with improved trajectories. In *2013 IEEE International Conference on Computer Vision*, pages 3551–3558, 2013.

[80] Jiangliu Wang, Jianbo Jiao, Linchao Bao, Shengfeng He, Wei Liu, and Yun-Hui Liu. Self-supervised video representation learning by uncovering spatio-temporal statistics. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2021.

[81] Jiangliu Wang, Jianbo Jiao, and Yun-Hui Liu. Self-supervised video representation learning by pace prediction. In *European conference on computer vision*, pages 504–521. Springer, 2020.

[82] Jinpeng Wang, Yuting Gao, Ke Li, Yiqi Lin, Andy J Ma, Hao Cheng, Pai Peng, Feiyue Huang, Rongrong Ji, and Xing Sun. Removing the background by adding the background: Towards background robust self-supervised video representation learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11804–11813, 2021.

[83] Haozhe Xie, Hongxun Yao, Xiaoshuai Sun, Shangchen Zhou, Shengping Zhang, and Xiaojun Tong. Pix2vox: Context-aware 3d reconstruction from single and multi-view images. *CoRR*, abs/1901.11153, 2019.

[84] Dejing Xu, Jun Xiao, Zhou Zhao, Jian Shao, Di Xie, and Yueting Zhuang. Self-supervised spatiotemporal learning via video clip order prediction. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10334–10343, 2019.

[85] Xiang Yan, Syed Zulqarnain Gilani, Mingtao Feng, Liang Zhang, Hanlin Qin, and Ajmal Mian. Self-supervised learning to detect key frames in videos. *Sensors*, 20(23), 2020.

[86] Yuan Yao, Chang Liu, Dezhao Luo, Yu Zhou, and Qixiang Ye. Video playback rate perception for self-supervised spatio-temporal representation learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 6548–6557, 2020.

[87] Fisher Yu and Vladlen Koltun. Multi-scale context aggregation by dilated convolutions. In Yoshua Bengio and Yann LeCun, editors, *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*, 2016.

[88] Xiaohua Zhai, Avital Oliver, Alexander Kolesnikov, and Lucas Beyer. S^4l: Self-supervised semi-supervised learning. *CoRR*, abs/1905.03670, 2019.

[89] Richard Zhang, Phillip Isola, and Alexei A. Efros. Split-brain autoencoders: Unsupervised learning by cross-channel prediction. *CoRR*, abs/1611.09842, 2016.

[90] Bin Zhao, Maoguo Gong, and Xuelong Li. Hierarchical multimodal transformer to summarize videos, 2021.

[91] Bolei Zhou, Aditya Khosla, Agata Lapedriza, Aude Oliva, and Antonio Torralba. Learning deep features for discriminative localization. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2921–2929, 2016.

[92] Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A. Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks. *CoRR*, abs/1703.10593, 2017.