Master's Thesis

AutoDeepFuse: Neural Architecture Search for Multi-Modal Stream Fusion in Deep Video Action Recognition

---

Jin Woo Ahn

Examiner: Prof. Frank Hutter, Prof. Thomas Brox
Advisers: Arber Zela, Mohammadreza Zolfaghari

Albert-Ludwigs-University Freiburg
Faculty of Engineering
Department of Computer Science
Chair for Machine Learning

March 12nd, 2021

Writing period

22. 10. 2020 – 12. 03. 2021

Examiner

Prof. Frank Hutter, Prof. Thomas Brox

Advisers

Arber Zela, Mohammadreza Zolfaghari

Declaration

I hereby declare, that I am the sole author and composer of my thesis and that no other sources or learning aids, other than those listed, have been used. Furthermore, I declare that I have acknowledged the work of others by providing detailed references of said work.
I hereby also declare, that my Thesis has not been prepared for another examination or assignment, either wholly or excerpts thereof.

Seoul, 10,03,2021

Place, Date

Signature

Abstract

In the field of deep video action recognition, approaches that leverage multi-modal information of the dataset, e.g., RGB, optical flow and pose estimate, have shown great success in recent years due to their ability to combine spatial and temporal information for more effective action recognition. However, while the performance of these approaches highly depend on the fusion scheme of the multi-modal streams, previous works have achieved this by simple methods in which the multi-modal information is fused at the final stage of the architecture, or by manually designing the fusion scheme of the streams at different layers.

In this thesis, we propose a novel approach, which we dub **AutoDeepFuse**, to the fusion of multi-modal streams via a 3D spatio-temporal CNN architecture. Moreover, instead of manually designing the architecture, we utilize PC-DARTS, a differentiable architecture search method, to automatically discover an optimal fusion network. Through our experiments, we show that our method is able to discover fusion networks that are compact in size and perform superior to the baseline fusion scheme that concatenates the modality stream at the final stage of the network. Furthermore, we contribute to a better understanding of PC-DARTS by studying the effect of various design considerations of PC-DARTS on the quality of the networks that it discovers. Through the analysis of the experimental results, we show that the search space design of PC-DARTS has a significant impact on the resulting performance of the found network, and accordingly, we provide suggestions on constructing an effective search space of PC-DARTS.

Contents

List of Figures

List of Tables

1 Introduction

Video action recognition, the task of understanding human actions in video, is a crucial research area in the field of computer vision with various real-world applications, including motion analysis, human-computer interaction, healthcare monitoring, security and gaming [Bloom, 2015, Antoshchuk et al., 2018, Ben Youssef et al., 2016, Gul et al., 2020]. With the recent success of convolutional neural networks (CNN) in image recognition, there has been a growing interest in the research community in applying well-performing CNN architectures to the task of video action recognition [Karpathy et al., 2014, Tran et al., 2015, Simonyan and Zisserman, 2014]. However, successfully classifying human actions has long been a challenging task. Contrary to object recognition in images, action recognition requires understanding of both the appearance and motion information from the 3D data whose volume is significantly larger than that of images. Further, the temporal nature of the data poses an additional difficulty of modelling long-range interactions between the frames. Hence, an action recognition algorithm must be able to take full advantage of the multi-modal information in the data in order to effectively recognize actions.

In the recent years, approaches incorporating multiple modalities of the data have been a popular option for addressing this issue. These approaches made use of various modalities of the data in which different aspects of the information are more explicitly captured. Simonyan and Zisserman [2014] pioneered in the development of this approach, where they used two separate CNNs for extracting the spatial (appearance) and temporal (motion) information from the RGB and optical flow of the data. Inspired by the success of their method, subsequent researches focused on extending the multi-stream methods to utilize other input modalities such as human pose estimate [Zolfaghari et al., 2017, Azar et al., 2018] and depth image [Roitberg et al., 2019].

In adopting multi-stream approach, a crucial aspect to consider is how to fuse the information extracted from each modality. Previous works that use multi-stream networks typically combined the extracted information at the very last stage by simply averaging the class scores of each stream output [Azar et al., 2018, Simonyan

and Zisserman, 2014, Yue-Hei Ng et al., 2015], or training a simple classifier such as an SVM [Simonyan and Zisserman, 2014]. However, these fusion methods are suboptimal, since the spatial and temporal information of the data at the pixel-level is not communicated between the streams, and therefore the networks are not able to recognize "what" is moving "where". To tackle this problem, later works experimented with more complex fusion methods by manually designing fusion scheme of the streams at different layers [Feichtenhofer et al., 2016, Park et al., 2016], or via a Markov chain [Zolfaghari et al., 2017]. The performance improvement brought by these methods indicate that effective fusion of data modalities is a crucial factor in building a multi-stream video action recognition architecture.

In this thesis, we propose a novel approach, which we dub **AutoDeepFuse**, to the fusion of multi-modal streams via a 3D spatio-temporal convolutional fusion architecture. Specifically, our approach uses a three-stream architecture, in which each stream extracts features of the RGB, optical flow and pose estimate of the data using pre-trained models and fuse the features via a 3D CNN. Through our experiments, we show that using a CNN to fuse the streams yields superior performance in comparison to the simple feature concatenation at the last stage of the architecture.

Further, instead of manually designing the fusion scheme, we propose to utilize PC-DARTS [Xu et al., 2019], a variant of differentiable architecture search (DARTS) [Liu et al., 2018b], to automatically discover an optimal network for multi-stream fusion. PC-DARTS is a gradient-based architecture search method that has been shown capable of finding network architectures that outperform architectures manually designed by human experts in various problem domains. Moreover, because PC-DARTS searches for an optimal architecture using gradient descent, it is very fast. In our work, we show that PC-DARTS is able to discover good fusion architectures in only 2 GPU days, thereby avoiding the cumbersome trial-and-error process of manually designing the multi-stream fusion scheme.

Our work also contributes to the DARTS research in several aspects. First, we adopt DARTS, which has been so far mostly applied to the task of image classification, to the more challenging task of video understanding, and show that DARTS is able to handle video data.

Further, we contribute to a better understanding of DARTS by studying the effect of various settings of DARTS on the quality of the architectures that it discovers. While numerous researches report DARTS to be successful, several other works also report that it does not perform well [Yu et al., 2019, Li and Talwalkar, 2020], the possible reasons for which have been investigated in follow-up works [Zela et al., 2019,

Chen et al., 2020]. There are two main drawback of DARTS: first, the architecture search is performed by training an over-parametrized *super-network*, but the optimal architecture is obtained by pruning the connections in the super-network to derive a *sub-network*. This discretization step introduces a discrepancy between the super-network and the sub-network, also known as the *optimization gap*, that may lead to performance deterioration. Second, while DARTS performs a micro-search, i.e., searching only for parts of a larger fixed network, on a small pre-defined search space, there are various *macro-level* search space designs that are to be manually determined by the implementer. Thus, when it is poorly designed, the search may result in a suboptimal architecture. In order to investigate these issues, we conduct a set of systematic experiments on various architecture search settings of PC-DARTS and provide valuable insights into the effects of each design considerations.

To summarize, the contributions of our work are as follows:

- We propose a novel approach to the fusion of multi-stream networks for the task of video action recognition, in which we make use of a 3D spatio-temporal convolutional fusion network.
- We employ differentiable architecture search to automatically find an optimal fusion network. We show that the architecture search is fast and efficient, and that the found network yields satisfying performance and is compact in size.
- We study and analyze the effects of various macro-level architectural designs on the performance and model size of the architecture found by PC-DARTS.

The structure of this thesis is designed as follows: first, we start with Chapter 2, where we describe the previous related works that motivated our research. Next, in Chapter 3, we elucidate the DARTS and PC-DARTS frameworks which form the basis of our architecture search method. Afterwards, we describe the components of the AutoDeepFuse pipeline, details of the fusion architecture search and evaluation procedure, and the architectural settings of PC-DARTS, that are experimented in this work, in Chapter 4. In Chapter 5, we explain the experimental procedure, and report and analyze the obtained results in detail. Lastly, we summarize and conclude our work in Chapter 6.

2 Related Work

2.1 Neural Architecture Search

Deep neural networks have achieved tremendous success on various problem domains in recent years [Collobert et al., 2011, Krizhevsky et al., 2012, He et al., 2016]; however, since the optimal architectural design for a specific task or dataset often differs significantly from another, this achievements have largely been the result of time-consuming trial-and-error process of manually adjusting the architecture and its hyperparameters by domain experts. To avoid this nuisance, Neural Architecture Search (NAS) [Elsken et al., 2018], the process of automatically designing well-performing neural network architectures, has recently gained interest in the deep learning community. NAS is capable of producing architectures superior to manually designed state-of-the-art architectures on domains such as image classification and language modelling [Zoph and Le, 2016, Real et al., 2017], and architectures on par with handcrafted architectures but with significantly less number of parameters [Liu et al., 2018b].

2.1.1 RL & EA

A variety of NAS strategies have been studied over the last years. Zoph and Le [2016] used reinforcement learning to train a recurrent neural network that produces well-performing convolutional and recurrent architectures for image classification and language modelling. Evolutionary strategies, the concept of evolving a pool of candidate architectures by letting the fittest survive and mutate over time, have also shown great success in image classification [Real et al., 2017].

2.1.2 Bayesian Optimization

Bayesian optimization (BO) in the context of NAS can be viewed as an iterative search strategy where the most promising candidate architecture configuration at a certain time step is selected based on the current objective function and evaluated to update the objective function for the next step [Shahriari et al., 2015]. Liu et al.

[2018a] used such strategy for the architecture search, using a surrogate model for fast performance prediction of a given configuration. Kandasamy et al. [2018] and Ru et al. [2020] employed a Gaussian-process-based BO architecture search combined with Weisfeiler-Lehman kernel and OTMANN metric, respectively. Zela et al. [2018] combined BO with Hyperband [Li et al., 2017] for the joint optimization of the architecture and its hyperparameters by efficiently allocating resources for evaluating configurations based on their expected performance.

2.1.3 One-Shot Approaches

While producing competitive results, the above methods suffer from extremely high computational cost, since they require (complete or partial) evaluation of potentially thousands of architectures which typically demands up to hundreds of GPU days. To alleviate this drawback, a recent line of researches focused on *one-shot architecture search* [Liu et al., 2018b, Pham et al., 2018, Brock et al., 2017], a search method in which a large over-parametrized meta-architecture that contains all possible instances of the search space is trained once to obtain the best architecture instead of training each instance separately. These methods have shown to be capable successful architectures with the search cost reduced by several orders of magnitude.

One particularly popular one-shot NAS method is differentiable architecture search (DARTS) [Liu et al., 2018b]. DARTS is a NAS framework in which the discrete search space is relaxed by assigining continuous architecture parameters to the discrete candidate operators in the meta-architecture, and the architecture parameters and model parameters are jointly optimized via gradient descent, achieving competitive performance while significantly reducing the search cost down to a few GPU days on image classification and language modelling. Inspired by the success of DARTS, several follow-up works have been conducted. GDAS [Dong and Yang, 2019] introduced a trainable sub-graph sampler to guide the search and improve search stability. Progressive DARTS [Chen et al., 2020] studied the phenomenon of optimization gap that occur at the discretization step of DARTS and ways to relieve it. PC-DARTS [Xu et al., 2019] reduced the memory cost of DARTS by sampling a subset of channels in the search. Our work is based on DARTS and PC-DARTS, and the details of both frameworks are elaborated in chapter 3.

2.2 Video Action Recognition

2.2.1 2D Frame Fusion

Inspired by the success of deep CNNs on visual data processing, previous works focused on extending 2D convolutional networks to additionally account for the temporal information. Based on the assumption that good representations of both spatial (appearance) and temporal (motion) features are pivotal for effective video action recognition, earlier works attempted to achieve it by first extracting spatial features from each frame and fuse them to learn temporal information. Karpathy et al. [2014] experimented with several frame fusing schemes, i.e., early, late and slow fusions. However, their results showed that these approaches performed significantly worse than existing models using handcrafted features, and that their approaches only slightly outperformed the single-frame baseline architecture that simply averages the predictions of each frame, thus concluding that the proposed architectures learn spatial features but do not adequately capture motion information.

Subsequent works have attempted to address this issue in various ways. Donahue et al. [2015] adopted a two-stage architecture where spatial features of individual frames are extracted first using 2D convolutions and then passed into LSTM [Hochreiter and Schmidhuber, 1997], a recurrent sequence learning module, for final prediction.

Similarly, Sun et al. [2015] introduced Factorized spatio-temporal Convolutional Netowrk ($F_{st}CN$), in which inputs are passed to 2D convolutional layers at earlier stages and then passed to two separate branches of further 2D spatial convolutions and 1D temporal convolutions, respectively, the outputs of which are combined at the end.

2.2.2 3D Spatio-Temporal Networks

Later works attempted to jointly learn spatio-temporal features from the data by using 3D convolutional operators. Tran et al. [2015] experimented with such 3D convolutional networks (C3D) and concluded that networks with small 3 x 3 x 3 (XYT) kernels perform well on action recognition tasks. Extending successful image classification architectures with 3D kernels has also been popular. Carreira and Zisserman [2017] and Hara et al. [2017] have respectively extended InceptionNet [Szegedy et al., 2015] and ResNet [He et al., 2016] by inflating 2D kernels to 3D. Further, Tran et al. [2018] proposed R(2+1)D, which uses the ResNet structure but replaces 2D convolution with a special type of 3D convolution that is factorized into 2D spatial convolution and 1D temporal convolution.

2.2.3 Multi-Stream Networks

Strategies involving multiple networks streams, in which each stream processes multiple input modalities of the data, have also been studied. Instead of capturing motion information from the RGB images alone, these approaches decomposed spatial and temporal components at the data-level by additionally using optical flow information. Simonyan and Zisserman [2014] experimented with a two-stream convolutional network architecture that extracts spatial and temporal features from raw RGB data and multi-frame optical flow, respectively, and combines the predictions of the two streams by averaging or training an SVM.

Multi-stream networks using different input modalities other than optical flow have also been explored. Azar et al. [2018] trained a four-stream network on different modalities: RGB, human pose estimation, optical flow and warped optical flow that removes camera motion, for group activity recognition tasks. Pose estimation has shown to enhance activity recognition, as it contains both spatial (i.e., location of body parts) and temporal (i.e., movement of body parts across frames) information that helps the model discriminate between actions. For recognition tasks such as video-based hand gesture recognition, where appearance information arguably plays a heavier role than motion information, depth images on top of RGB images have been used [Roitberg et al., 2019]

2.2.4 Multi-Stream Fusion

While previous works have shown that multi-stream architectures are able to significantly improve the recognition performance, these architectures typically either fused the class scores of each stream or fused the final feature maps from each stream via addition or concatenation at the last stage of the architecture, limiting the network's capability to learn joint spatio-temporal information at intermediate layers. Thus, the architectures were not able to effectively learn actions that occur at a specific pixel and time, which may be crucial in discriminating between similar actions (e.g., brushing hair and teeth).

To address this issue, several follow-up works experimented with more complex fusion schemes and showed that effective stream fusion is necessary for further improvement in performance. Roitberg et al. [2019] used cross-stitch units that enables information flow between the streams at every layer. Feichtenhofer et al. [2016] considered various ways to fuse the features, including addition, concatenation and 1x1 convolution, of the multi-streams networks to at different layers such that

channel responses at the same pixel position are put in correspondence. Later, they experimented with gating units to learn where the networks should fuse the information across the streams [Feichtenhofer et al., 2017].

2.3 NAS on video action recognition

Given the promising results of NAS in the area of image classification, as well as competitive performance of manually designed architectures for video understanding, the next logical step would be to combine the two fields. However, little attempts have been made so far to apply NAS on video action recognition tasks due to the challenge of building efficient operations for capturing spatio-temporal information as well as larger computational cost. Currently existing approaches are described below.

Piergiovanni et al. [2019] developed an evolutionary algorithm in which a pool of networks consisting of a fixed number of blocks are evolved over time. Each block in a network is initialized to be an operation or a module consisting of multiple operations combined in an InceptionNet-like manner and has a residual connection to the next block. The number, type and connectivity of the operators inside the blocks are then optimized via mutation and evolution.

Similarly, Ryoo et al. [2019] proposed an evolutionary strategy that finds the optimal multi-stream architecture that receives RGB frames and optical flow images at different temporal resolutions as inputs. The network consists of four layers, each containing multiple parallel blocks (i.e., sub-networks consisting of 2D and (2+1)D convolutions). A block in the network may receive outputs from any other blocks from the previous layers, and likewise, provides its output to any other blocks of the later layers. The connectivity as well as the channel size and the temporal resolution (controlled with dilated 1D convolution) of each block is learned via evolution.

Another work which is closely related to ours is by Peng et al. [2019]. The authors employ DARTS to search for the best-performing modules for video action recognition, using RGB frames as the input. This approach has yielded performance superior to the manually designed architectures, while significantly reducing the computational cost compared to the evolutionary strategies.

Our work also utilizes differentiable architecture search to discover optimal action recognition architecture; however, our approach differs from that of Peng et al. [2019] in that 1) instead of searching for a classification network that takes only RGB frames as the input, we search for a fusion network that takes multiple modalities of the data, i.e., RGB, optical flow and pose estimate, for the action recognition, and that

2) we use the features of the modalities extracted from pre-trained models for the architecture search, instead of directly providing the modalities as the input, to improve the performance.

3 Background

In this chapter, we describe DARTS [Liu et al., 2018b] and its extension, Partially-Connected DARTS (PC-DARTS) [Xu et al., 2019], and define notations used throughout this thesis.

3.1 DARTS

The core idea of DARTS is to continuously relax the discrete architecture search space of operators and connectivity patterns of layers by introducing continuous architectural parameters that represent the strength of each connection and operation, enabling efficient end-to-end architecture search via gradient descent. The DARTS procedure consists of two phases: the search phase, in which the optimal architecture is searched, and the evaluation phase, where the found architecture is trained from scratch and evaluated. The details of each phase are elaborated below.

3.1.1 Architecture Search

DARTS decomposes the network architecture into repeating blocks of sub-networks, also known as cells. A cell is a directed acyclic graph (DAG) consisting of a sequence of input nodes, N hidden nodes, and an output node, where every hidden node is connected to all of its predecessor nodes. Each node $x^{(i)}$ is a latent representation, and each edge (i, j) is associated with a pre-defined set $\mathcal{O}$ of candidate operations $o^{(i,j)}$ that transforms $x^{(i)}$. A candidate operator may have parameters w (e.g., convolution), or be parameter-less (e.g., skip-connection and *zero*, representing lack of connection). For each hidden node $x^{(j)}$, its value is computed as follows:

$$x^{(j)} = \sum_{i<j} o^{(i,j)}(x^{(i)}), \tag{1}$$

where $0 \leq i < j \leq N$ and $x^{(i)}$ are the predecessor nodes of $x^{(j)}$. The output of a cell is computed by taking the summation of all values of the hidden nodes. Instead of using

discrete operations, DARTS continuously relaxes the search space by formulating $o^{(i,j)}$ as a weighted sum of all candidate operations:

$$\bar{o}^{(i,j)}(x) = \sum_{o \in O} \frac{\exp(\alpha_o^{(i,j)})}{\sum_{o' \in O} \exp(\alpha_{o'}^{(i,j)})} o(x), \tag{2}$$

where $\alpha_o^{(i,j)}$ is the architectural parameter that determines the strength of operation o associated with the edge (i, j). In other words, each edge is associated with a mixed operation that takes the weighted sum of the outputs of each operation. With this formulation, gradients can be propagated through α, allowing the architecture to be optimized in a fully differentiable manner.

The continuous relaxation thus transforms the search problem as the task of learning optimal architectural parameters α^* for determining the best operator for each edge. The goal of DARTS is therefore to find α^* that minimizes the validation loss L_{val} of the given dataset:

$$\min_{\alpha} L_{val}(w^*(\alpha), \alpha), \tag{3}$$

where $w^*(a)$ denotes the optimal network parameters with respect to the training dataset under the given α. Because L_{val} depends both on α and w, this formulation gives rise to a bi-level optimization problem, in which the network parameters need to be optimized repeatedly for every update step of the architecture parameters. The naive approach, in which the network parameters are trained until convergence at each step, would be intractable. Instead, the authors propose to approximate $w^*(a)$ with only one gradient update step:

$$\begin{aligned} &\nabla_\alpha L_{val}(w^*(\alpha), \alpha) \\ &\approx \nabla_\alpha L_{val}(w - \xi \nabla_w L_{train}(w, \alpha), \alpha) \\ &= \nabla_\alpha L_{val}(w', \alpha) - \xi \nabla^2_{\alpha,w} L_{train}(w, \alpha) \nabla_{w'} L_{val}(w', \alpha), \end{aligned} \tag{4}$$

where w and ξ denote the current network parameter value and the learning rate, respectively, and $w' = w - \xi \nabla_w L_{train}(w, \alpha)$. While the convergence guarantees of DARTS using this one-step approximation is yet to be proven, the authors empirically demonstrate it converges to a stable point with an adequate value of ξ. The approximation can be viewed as a trade-off between the computational cost and performance. Moreover, the cost can be further reduced by omitting the expensive

computation of the second-order gradient, i.e., setting $\xi = 0$, at the cost of tolerable amount of loss in accuracy. The resulting first-order approximation $\nabla_\alpha L_{val}(w, \alpha)$ can be viewed as assuming that the current network parameter w is equal to $w^*(\alpha)$ at every architecture parameter update step.

3.1.2 Evaluating the found Architecture

After the search phase, the final cell is constructed by discretizing the operations in each edge. The discretization process is done as follows: for each hidden node j, we consider the soft-maxed α values of the operations between j and its predecessor nodes, i.e., $\frac{\exp(\alpha_o^{(i,j)})}{\sum_{o' \in O} \exp(\alpha_{o'}^{(i,j)})}, 0 \leq i < j$. Then, we retain the top-K operations of each node with the largest values. In the original paper, K is set to 2, but other discretization strategies may be used, such as retaining $K = 3$, or all operations, which is an aspect we study in this work (see section 4.2.3). Once the optimal cell is found, the final network is constructed by stacking the found cells. Finally, the derived network is trained from scratch to evaluate its test performance.

3.2 PC-DARTS

Despite the significant improvement in runtime, DARTS still suffers from large memory requirements, since the intermediate outputs of all operations in each node have to be stored while training the search architecture. This problem puts a limit on searching with a large batch size, leading to slower search and instable training. To address this issue, Xu et al. [2019] proposed an extension of DARTS that samples a subset of channels and apply operations only on the subset while leaving other channels intact. Mathematically, the idea is described as follows. Let x_i be a latent video representation of size $[H, W, T, C]$ that undergoes a transformation, where H, W, T are the height, width and number of frames of the input, respectively, and C is the number of channels. The computation of the partial channel connection is given by

$$o_{PC}^{(i,j)}(x^i) = \sum_{o \in O} \frac{\exp(\alpha_o^{(i,j)})}{\sum_{o' \in O} \exp(\alpha_{o'}^{(i,j)})} o(S^{(i,j)} * x^i) + (1 - S^{(i,j)}) * x^i, \tag{5}$$

where $S^{(i,j)}$ denotes the mask that selects the channels to keep. For each mixed operation, $1/K$ of the channels are selected at random, where K is a hyperparameter controlling the proportion of selected channels.

The main benefits of partial channel sampling in DARTS is twofold. First, the memory requirement of the computation is reduced to $1/K$, since only $1/K$ of the channels are passed to the mixed operations. As a result, larger batch sizes can be used for training with increased speed and stability. Second, partial channel sampling regularizes the preference of parameter-less operations. An often reported undesired phenomenon in DARTS is that parameter-less operations, such as $skip-connect$, $zero$, $pool$, or even operations such as $noise$ that are deliberately designed to harm the performance, are preferred over operations with parameters such as convolution [Zela et al., 2018]. This is presumably because parameter-less operations are heavily preferred at earlier stages of the architecture search, since their outputs are more stable than those of operations whose parameters are not yet well-trained, to the point where operations with parameters are not able to catch up even at later stages. Partial channel sampling mitigates this problem because operations with parameters produce stabler results as only a small subset of the channels are passed through under-trained operations while others remain intact.

However, randomly sampling channels may still cause suboptimal operation selection, since the subset of channels with which the parameters are trained change at every iteration. As a result, the final optimal architecture could vary widely. To compensate for this randomness, the authors introduce a set of edge normalization parameters β associated with each edge (i, j). $\beta^{(i,j)}$ represents the relative strength of the edge (i, j) in comparison to the other edges that are connected to the same node, i.e., $(i', j),\ i' < j$. The computation of the value of a hidden node j is as follows:

$$x_{PC}^{j} = \sum_{i<j} \frac{\exp(\beta^{(i,j)})}{\sum_{i'<j} \exp(\beta^{(i',j)})} o^{(i,j)}(x^i) \tag{6}$$

β parameters are jointly optimized with α and are shared across iterations, thereby rendering the final architecture less influenced by random sampling at each iteration. After the search is finished, the final archictecture is derived based on the operation strength given by $\alpha_o^{(i,j)} \cdot \beta^{(i,j)}$.

4 Approach

In this chapter, we propose our approach, dubbed **AutoDeepFuse**, to the problem of video action classification.

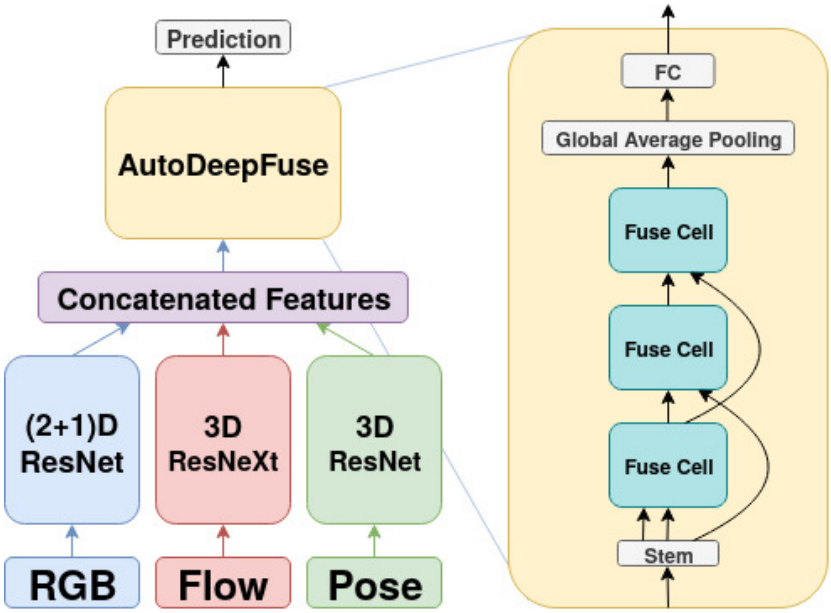

Figure 1: The AutoDeepFuse framework. Multi-modal features are first processed using pre-trained models and then fused via fusion architecture searched through differentiable architecture search.

4.1 AutoDeepFuse Framework

The AutoDeepFuse pipeline consists of two components: 1) extracting features of each modality, i.e., RGB, optical flow and pose estimate of the data, using models pre-trained on Kinetics700, and 2) combining the extracted modality features through the fusion architecture found via architecture search to create predictions for the video action recognition task. In this section, we present a detailed overview of each component. Specifically, we elucidate the input modalities and their respective

pre-trained models used for our experiments, and explain the components of the AutoDeepFuse search block in which the architecture search is performed.

4.1.1 Input Modalities

Optical Flow

For successful recognition of actions in videos, good representations of both appearance and motion information are necessary. While 3D CNNs are in principle able to learn both representations solely from raw RGB data, researches have shown that explicitly providing different input modalities that capture appearance and motion information yields superior performance [Simonyan and Zisserman, 2014]. Accordingly, we use the optical flow images of each frame extracted from the RGB data for our experiments.

An optical flow encodes the apparent motion of objects through a set of displacement vectors that describe the movement of pixels between two consecutive frames. Formally, let u, v be the pixel position of an image along the x and y axis, respectively. Then, an optical flow image consists of displacement vectors $\mathbf{d}_t(u, v) = [\mathbf{d}_t^x, \mathbf{d}_t^y]$ for each pixel (u, v) between two consecutive frames t and $t + 1$, so that the position of the object at pixel $(u, v)_t$ at frame t shifts to the pixel $(u, v)_t + \mathbf{d}_t$ at frame $t + 1$. $\mathbf{d}_t^x$ and $\mathbf{d}_t^y$ are the vertical and horizontal components of the displacement vector. For this work, we use the method by Zach et al. [2007] to extract the vertical and horizaontal components of the displacement vectors and stack them to form a 2D image for each frame.

Pose

Additionally, we use body pose information in the form of human body segmentation as an input modality. Explicitly providing the pose as well as the location of each body part of the human actor in a video is beneficial for action recognition in several ways. First, it allows the model to better detect parts of the image that belong to the human actor which might be harder to achieve solely from the RGB data. Also, the location information provides context to the motions that occur in a video, aiding the model's ability to distinguish whether a moving part of a video is a body part or an object. Such information can be especially helpful in videos where the actor interacts with an object (e.g. in a scene where a person throws a ball).

For this work, we extract the body pose estimate in the form of body parr segmentation for each frame of the videos using the method from [Zolfaghari et al., 2017], who trained the Fast-Net architecture [Oliveira et al., 2016] for the body part

segmentation task on the J-HMDB [Jhuang et al., 2013] and the MPII [Andriluka et al., 2014] datasets and achieved good performance. As in the original paper, we convert each segmented frame into an RGB image, in which each pixel located at a specific body part is assigned a pre-defined RGB value.

4.1.2 Pre-trained Models

In searching for a video action recognition architecture, one often uses the modalities directly as input to the search algorithm, as is done by Peng et al. [2019] and Ryoo et al. [2019]. Instead, we first perform feature extraction of the input modalities through the use of pre-trained models. For each input modality, we extract its features by passing it into its respective pre-trained model whose weights are trained to classify images in Kinetics700 [Carreira and Zisserman, 2017], a large video action recognition dataset containing approximately 650,000 video clips and 700 action classes. Because the pre-trained models are trained for the same task, i.e., human action recognition, the feature detectors learned in their intermediate layers also serve as effective detectors for our task. Moreover, since they are trained on a large dataset with 700 classes, the trained feature detectors are unlikely to overfit but remain generally applicable to other action recognition tasks. Hence, we expect the features extracted from the pre-trained models to serve better inputs to our fusion architecture than the modalities themselves.

For our experiments, we employ a variant of ResNet [He et al., 2016], the state-of-the-art architecture for image classification, for each input modality. For RGB, flow and pose inputs, we used (2+1)D ResNet [Tran et al., 2018] of depth 50, 3D ResNeXt [Xie et al., 2017] of depth 101, and 3D ResNet of depth 18, respectively. We obtain the network weights of the pre-trained models from the work of Kataoka et al. [2020] that trained a variety of spatio-temporal CNNs on the RGB and optical flow data of various datasets. Since the pre-trained model weights for the pose estimate is not available by Kataoka et al. [2020], we use the weights trained on RGB for our pose network.

In using pre-trained models for multi-modal fusion, a pivotal design decision is determining the layer at which to take the intermediate representation. A conventional spatio-temporal CNN is designed such that the spatial and temporal extent of the data is gradually reduced by increasing the stride value and/or the kernel size, or by introducing pooling layers. In principle, it is possible to use the output of any arbitrary intermediate layer of the pre-trained network to apply further fusion. In practice, however, the quality of the fusion varies significantly depending on which

layer the feature is taken from. If we use low-level features, i.e., features with large spatio-temporal resolution, as the input to our fusion architecture, we would not take full advantage of the benefits of using a pre-trained network since we only use its first few layers. On the other hand, taking high-level features from the very last layer would also be problematic, because the convolution operators in the fusion architecture require inputs with sufficiently large spatio-temporal resolution in order to perform effectively. Hence, our goal here is to select the layer of the pre-trained network that is sufficiently deep while its output resolution is large enough.

As illustrated in figure 2, all of the used pre-trained models contain four layer blocks that applies a series of convolution followed by max-pooling that reduces the spatio-temporal resolution by half. In our work, the outputs of the third layer of the pre-trained networks are used as inputs the fusion architecture, since they provide a good balance between the resolution size and the layer depth.

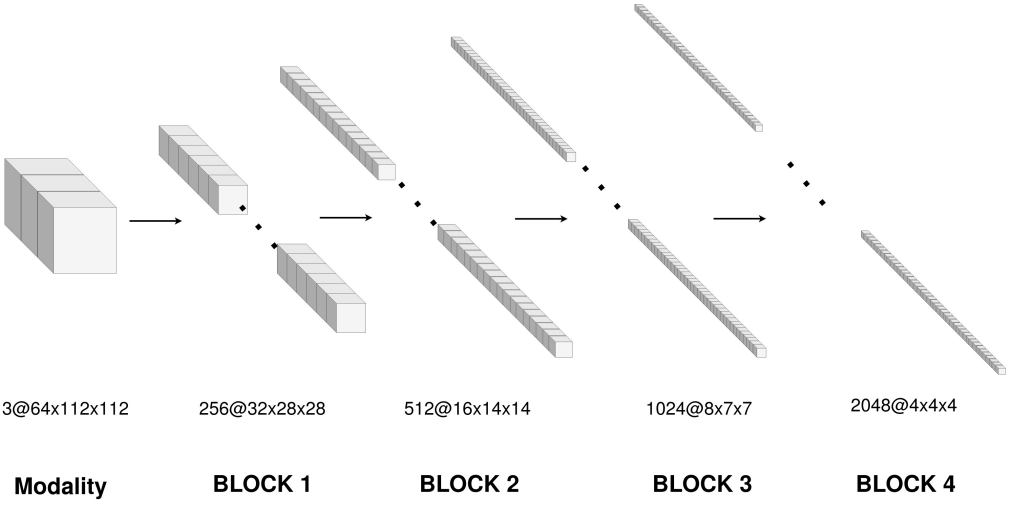

Figure 2: Transformation of feature map size $[C, T, H, W]$ in the pre-trained models, where C, T, H and W denote respectively the number of channels, temporal length, height and width of the feature maps.

4.1.3 Fusion Architecture

Using the features extracted from the three modality streams, we proceed to build a fusion network architecture. The building procedure is as follows: first, we perform architecture search using PC-DARTS [Xu et al., 2019] in order to derive the optimal network. Then, we train the resulting architecture from scratch. The components of

the fusion architecture as well as the details of the search and evaluation procedure are described below.

Stem Cell

Before performing the fusion, we first feed the features extracted from the pre-trained models to the stem cell $\mathcal{C}_{stem}$. To reduce the computational cost, $\mathcal{C}_{stem}$ concatenates the features and applies 1D convolution followed by batch normalization to reduce the channel size to 512.

Fuse Cell

Following the experiments in Liu et al. [2018b], we design our search space as a sequence of stacked cells $\mathcal{C}_t \in \{\mathcal{C}_1, ..., \mathcal{C}_n\}$ in which the features are transformed. Each cell is a directed acyclic graph containing input nodes and four hidden nodes, as illustrated in figure 3. The input nodes receive outputs of the closest previous cells that are connected to the current fuse cell and perform ReLU-Conv-BN to set the channel size to 512. Each hidden node of a cell is connected to all of its predecessor nodes (including both input and hidden nodes) via edges that represent mixed operation. During the search phase, each hidden node outputs the element-wise sum of all inputs, i.e., the outputs of its predecessor nodes transformed via mixed operations. In the evaluation phase, the cells are first discretized based on the optimized α values and trained from scratch. Each hidden node receives the outputs of the previous nodes that are connected to itself and applies the operator selected through the search. Formally, the output of a node j at the evaluation phase is:

$$x^{(j)} = \sum_{i<j} o^{(i,j)}(x^{(i)}) \qquad \text{if } \exists(i,j), \tag{7}$$

where $o^{(i,j)}$ is the operator of the edge (i, j) selected during the search.

The final output of the cell is the channel-wise concatenation of all hidden node outputs.

Final Prediction

After the fusion, we remove the spatial and temporal resolutions of the last fuse cell output by downsampling through the global average pooling layer. Afterwards, the downsampled features are fed to a fully-connected layer to produce the final prediction.

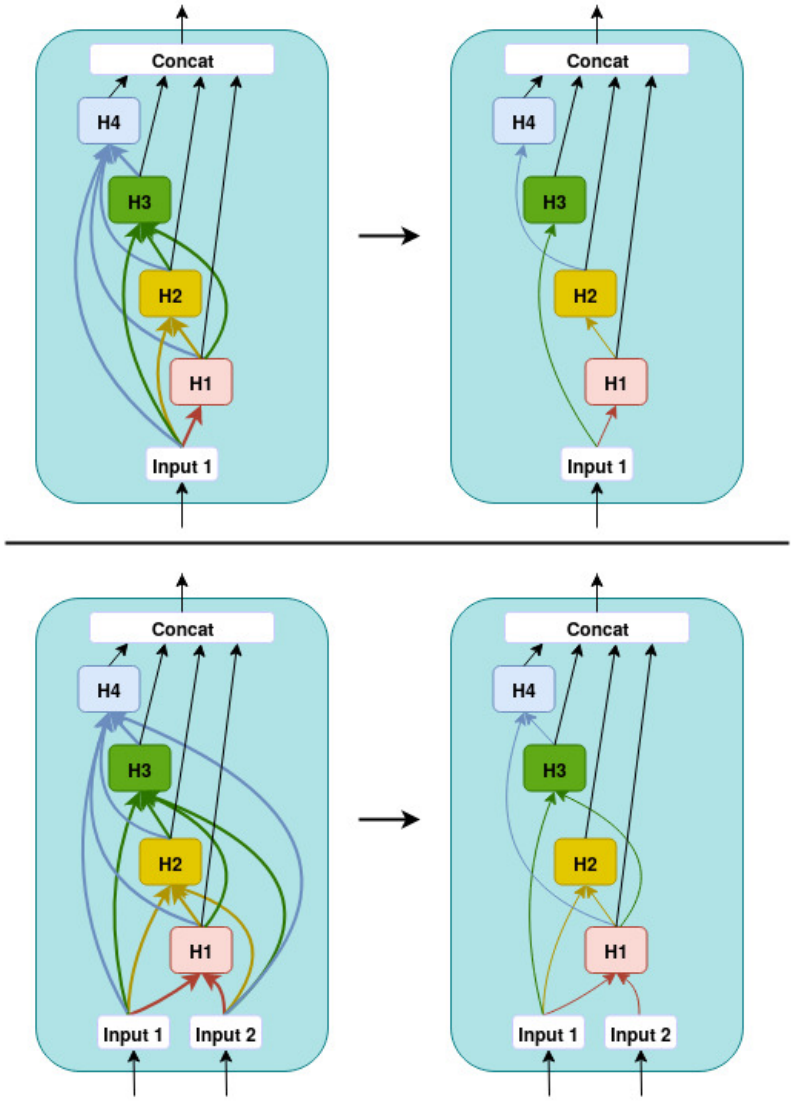

Figure 3: Fuse cell with $I = 1$ (top) and with $I = 2$ (bottom). During the search phase (depicted on the left), each hidden node receives outputs of previous nodes and applies mixed operation (bold colored arrows). After the discretization (on the right), each node receives outputs of previous nodes that are connected to itself and applies the operation chosen from the search. The cell output is the channel-wise concatenation of all hidden nodes outputs.

4.2 Architecture Search Design

Having elucidated the components of the AutoDeepFuse pipeline, we proceed to detail the various architecture design choices that are experimented in this work. Although differentiable architecture search is aimed at automatically discovering well-performing architectures for a given task, it is only able to perform architecture search on the cell level, i.e., operator choices and connectivity of nodes within a cell, while the macro-level decisions, such as the number of stacked cells to use and their connectivity pattern, are left to the implementer. Nevertheless, these decisions must not be overlooked since they are crucial for determining DARTS' behavior during the search and evaluation, the overall efficiency of the found architecture as well as its final performance. Hence, in order to conduct a thorough analysis of our approach, we run experiments on various settings of the macro-level design decisions.

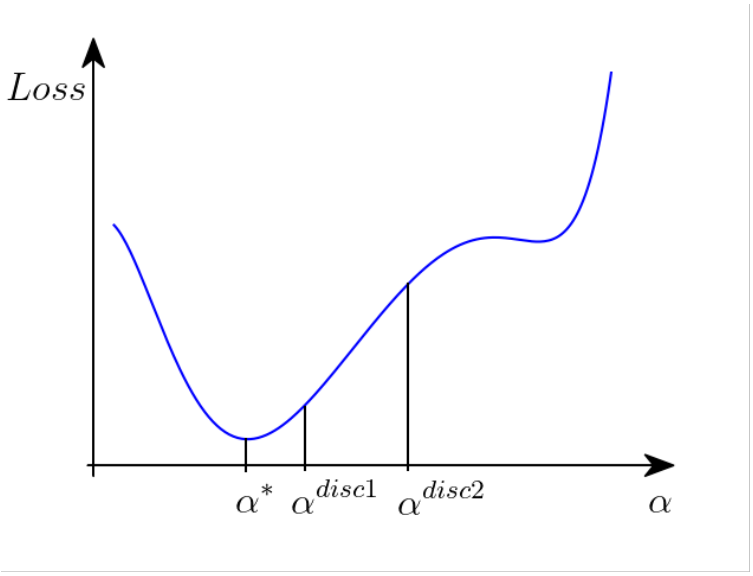

Figure 4: Hypothetical Loss function with respect to α. The further away the discrete architecture lies from the searched architecture, the higher the risk its performance is poor.

In investigating the effect of these decisions, we are primarily concerned with two aspects of the architecture search. First, we consider the **cost-performance trade-off** between the architecture search/evaluation procedure and the found network. Since the goal of our research is to discover an efficient fusion architecture block that is built on top of the pre-trained modality streams rather than a stand-alone classification network, we want to reduce the size of the searched architecture while maintaining

high performance. Second, we are concerned with reducing the **optimization gap** that occur at the discretization step of DARTS. Optimization gap is the discrepancy that arise when the optimized search architecture, expressed by the continuous α^*, is transformed to obtain the discrete architecture α^{disc} that lie somewhere in the neighborhood of α^*. If a discretization method produces α^{disc} that heavily distorts α^*, i.e., the distance between α^* and α^{disc} is too large, then the rankings of architecture candidates deemed by the search architecture would degrade to the point where they fail to reflect the true performance of the candidates, and as a result, the derived architecture would run the risk of performance deterioration (illustrated in figure 4). Hence, appropriate discretization strategy is necessary to avoid this problem. These considerations are reflected in our experimental design, described below.

4.2.1 Number of Cells

First, we study the relation between the depth of the fusion architecture and its performance. As is well known, deeper models have higher expressive capacity and therefore achieve higher performance when solving complex tasks. On the other hand, shallower models may achieve better generalization error since its lower expressive capacity inhibits overfitting to the given data. Further, shallow models are computationally less expensive and are easier to train. Since the aim of our research is to find a well-performing and efficient modality fusion architecture, we run experiments with varying numbers of stacked fuse cells C ranging from 1 to 4.

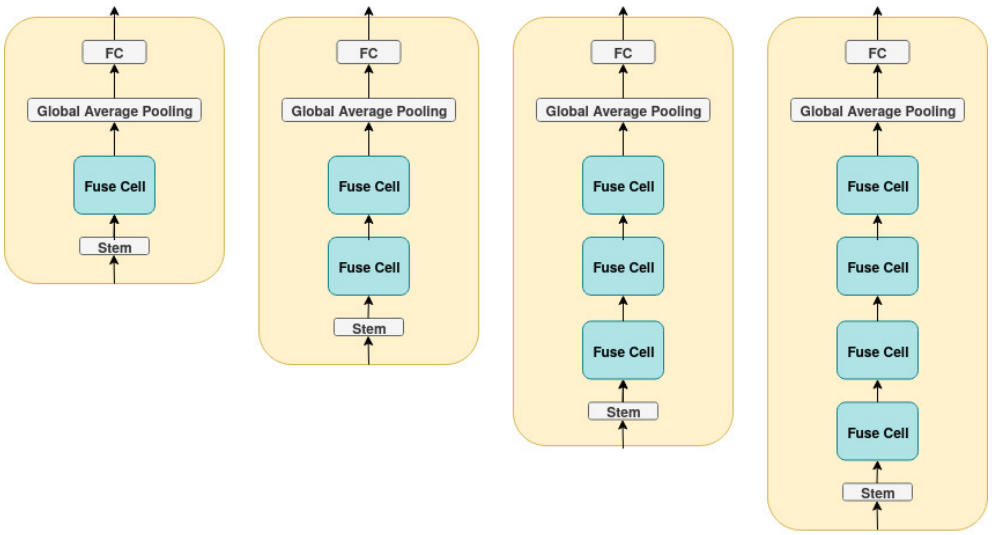

Figure 5: AutoDeepFuse search architecture with different number of fuse cells.

Our approach to stacking cells to construct the fusion architecture differs from the works by Xu et al. [2019] and Liu et al. [2018b] in two aspects. First, in the aforementioned works, the depth of the derived architecture is increased by repeating the discovered cells multiple times. For example, the architecture search of DARTS is done by stacking 6 search cells, while the evaluation is done by stacking the found cells to form a network of 20 cells (CIFAR-10) or 14 cells (ImageNet). However, arbitrarily increasing the network depth at the discretization step results in a large distance between α^* and α^{disc}, a phenomenon also known as the *depth gap* between the search and evaluation [Chen et al., 2020]. In this work, we avoid this problem by keeping the number of cells in the discretized network equal to that of the search architecture. Second, in the previous works, the architecture weights α (and the edge weights β in PC-DARTS) are shared across the same type of cells, i.e., normal and reduction cells, during the search. However, we believe that in the context of modality stream fusion, the features may require different types of operation at different level of layers, and therefore we train α and β of the cells independently, such that each cell at the end of the search contains different operators and node connections.

4.2.2 Cell connectivity

Further, we study the influence of cell-level connectivity on the performance by varying the number of a cell's input connections I from previous cells, as illustrated in figure 6. I determines the number of inputs a cell receives from its predecessor cells, i.e.,

$$out(\mathcal{C}_t) = \mathcal{C}_t(out(\mathcal{C}_{t-1}), ..., out(\mathcal{C}_{t-i})) \qquad i \in \{1, ..., I\}. \tag{8}$$

For example, when $I = 2$, the inputs to $\mathcal{C}_3$ are the outputs of $\mathcal{C}_2$ and $\mathcal{C}_1$, and the inputs to $\mathcal{C}_4$ are the outputs of $\mathcal{C}_3$ and $\mathcal{C}_2$, and so on. I also determines the total number of node connections M inside a cell, therefore also the computational cost of our algorithm. Since the number of hidden nodes in a cell is fixed as 4 throughout our experiments, a cell contains $M = 4(I-1)+10$ connections (see figure 3). In most differentiable architecture search applications, I is usually fixed to 2. In this work, however, we experiment with $I \in \{1, 2, 3\}$. When I is larger than 1, it is possible that a cell has less number of predecessor cells than I. In such case, the cell receives the $\mathcal{C}_{stem}$ output multiple times, as illustrated in figure 6.

The choice of I can be seen as a trade-off between cost and performance. Using

$I = 1$ makes the architecture to remain small, since there are less total number of edges to optimize in the cells. On the other hand, when $I = 3$, we can expect higher performance since the expressive capacity of the architecture is higher, and the dense connections permit information and gradients to propagate more easily.

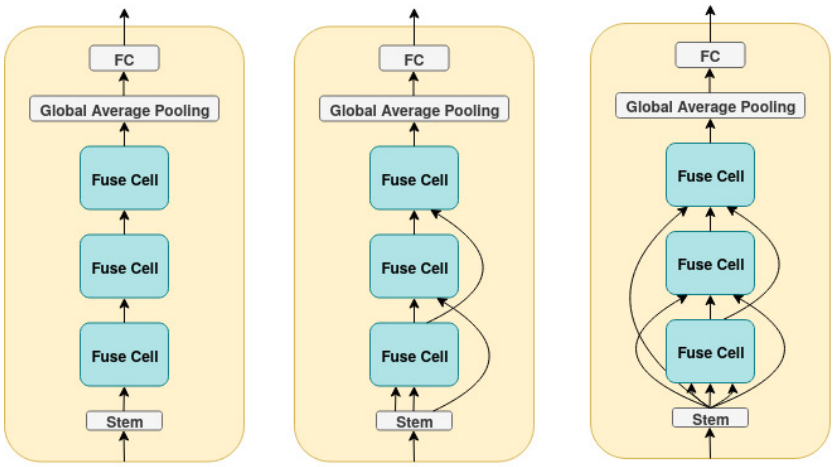

Figure 6: Fusion architecture with cell input connection $I = 1...3$ (from left to right).

4.2.3 Cell Discretization

We also experiment with different methods of discretizing cells for obtaining the final architecture. One method, as used in Xu et al. [2019], is to derive a cell by computing for each candidate operator $o^{(i,j)}$ its connection strength given by $\alpha_o^{(i,j)} \cdot \beta^{(i,j)}$ and keeping $K = 2$ operators with the largest connection strength per node (see section 3.2). We call this method type $top - K$ discretization. Another possible approach, which is not explored in previous works, is to retain an operator from all connections (i, j) inside a cell (we call this method type *all* discretization). The two methods are illustrated in figure 7.

The rationale behind the former approach is that only a subset of the optimized operators contribute mostly to solving the task, and therefore other operators can be removed to reduce the model size without harming the performance. However, we hypothesize that this could lead to poor performance, because as far as the problem of optimization gap is concerned, α_{disc} obtained through the $top - K$ method introduces more distortion of α_* than when the *all* method is used due to the arbitrary exclusion of operators trained during the architecture search phase. For example, the mixed

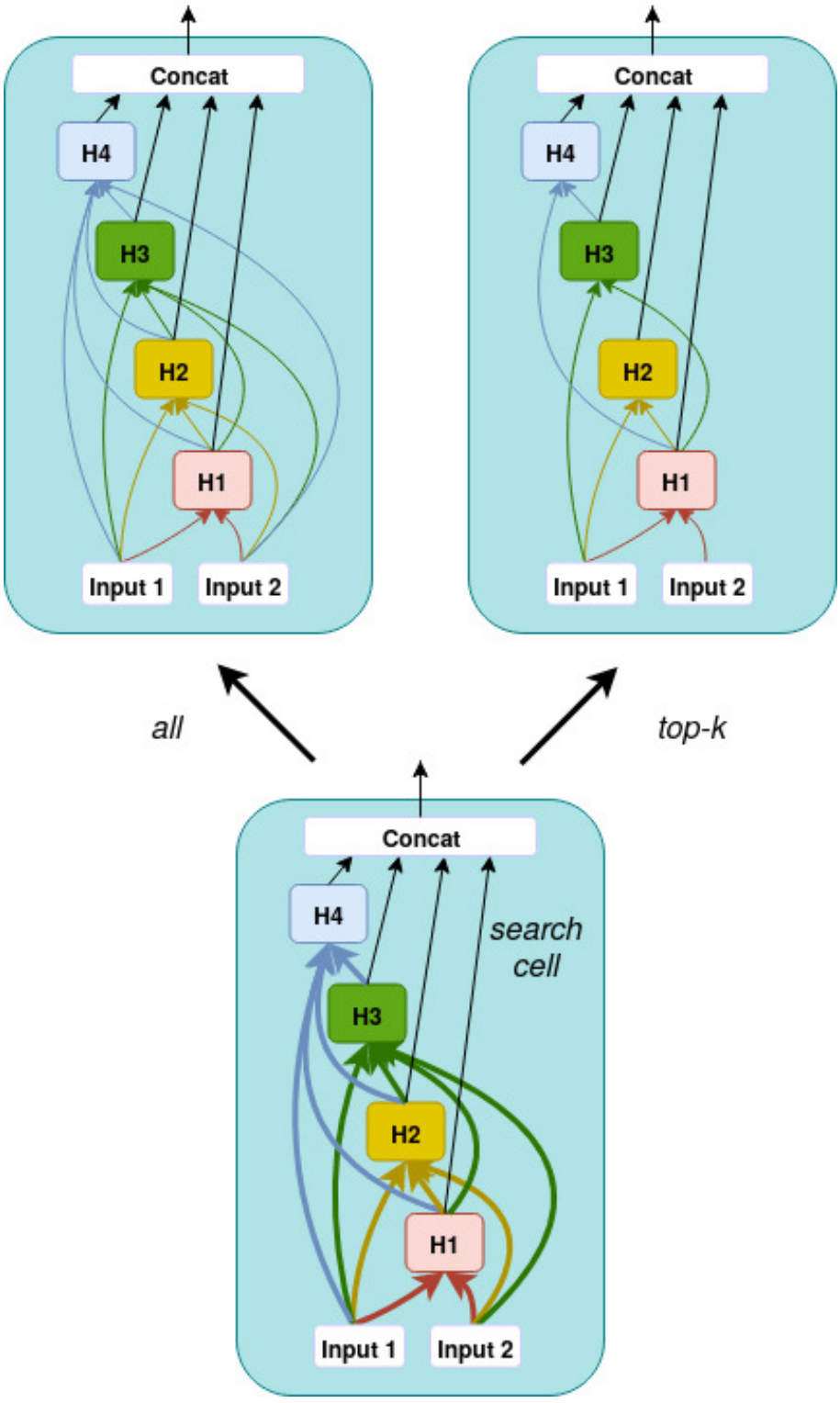

Figure 7: Fuse cell in the search phase (bottom), discretized cell of type *all* (top left) and type *top-k* (top right). Since $I = 2$ in the example, i.e., there are two inputs to the cell, all intermediate nodes retain two connections

operator of a hidden node might determine a certain candidate operator as the best choice based on the three inputs it receives during the search. However, it is possible that the operator is no longer the best choice if it only receives two of the inputs instead of all three. Then, the resulting network's performance would deteriorate because it is no longer a close approximation of α_*. The *all* discretization method can mitigate this problem by retaining all optimized edges, at the cost of higher computational resources.

To validate our hypothesis, we experiment with and compare the effect of both types of discretization. When the $top-K$ method is used, we set K to be equal to I, i.e., if a cell has three inputs, then three connections per node are retained.

4.2.4 Candidate Operators

Another architecture search design choice that we consider is the design of the set of candidate operators. For this work, we experiment with two sets of candidate operators. The first set, which we dub **S1**, includes operators conventionally used for video action recognition tasks: $\{3\times3\times3\ Conv, 1\times1\times1\ Conv, 3\times3\times3\ SepConv, 3\times3\times3\ DilConv, 3\times3\times3\ (2+1)D\_Conv, SkipConnect, Zero\}$. All operators maintain the same feature map size throughout the fusion by applying appropriate amount of $ZeroPadding$ around the feature maps. Further, the input and output channel dimensions of all operators are fixed to 512, and all operators except $SkipConnect$ and $Zero$ are preceded by Rectified Linear Unit layer and followed by batch normalization. $(2+1)D\_Conv$ [Tran et al., 2018] is a block of two layers comprising a 2D spatial convolution and a 1D temporal convolution, and therefore has intermediate feature maps between the layers. As in the original paper, we set the channel size of the intermediate feature maps to 1152, such that the overall number of parameters is equal to the 3D convolution with the same kernel size. The operators in **S1** and their parameter sizes are listed in table 1.

The second set **S2** is a modified set containing operators with less number of parameters that we created after observing DARTS's strong preference for parameter-heavy operators. Since our goal is to find an architecture block that efficiently fuses the already processed modality features, we experiment with **S2**, in which operators have less number of parameters, listed in table 2. Its difference to **S1** is that $3\times3\times3\ Conv$ and $3\times3\times3\ DilConv$ are excluded, the intermediate channel size of $(2+1)D\_Conv$ is reduced to 512, and $Flattened\_Conv$ [Jin et al., 2014] is included. Similar to $(2+1)D\_Conv$, $Flattened\_Conv$ consists of a series of three convolutional filters of size $(3\times1\times1)$, $(1\times3\times1)$ and $(1\times1\times3)$ that convolve

Name	Description	#Parameters
3D_Conv1	3D convolution with kernel size 1	0.26M
3D_Conv3	3D convolution with kernel size 3	7.08M
3D_SepConv3	3D depthwise convolution with kernel size 3 followed by 3D convolution with kernel size 1	0.55M
3D_DilConv3	3D convolution with kernel size 3 and dilation 2	7.08M
(2+1)D_Conv3	2D spatial convolution followed by 1D temporal convolution with kernel size 3	7.08M
Skip	Skip (identity) connection between nodes	none
Zero	No connection between nodes	none

Table 1: Candidate operators of **S1**. All operations are preceded by rectified linear unit (ReLU) and followed by a batch normalization layer.

Name	Description	#Parameters
3D_Conv1	3D convolution with kernel size 1	0.26M
3D_SepConv3	3D depthwise convolution with kernel size 3 followed by 3D convolution with kernel size 1	0.55M
(2+1)D_Conv3	2D spatial convolution followed by 1D temporal convolution with kernel size 3	3.15M
Flattened_Conv	A chain of 3D convolutions with kernel sizes (3, 1, 1), (1, 3, 1), (1, 1, 3)	2.36M
Skip	Skip (identity) connection between nodes	none
Zero	No connection between nodes	none

Table 2: Candidate operators of **S2**. All operators are preceded by rectified linear unit (ReLU) and followed by a batch normalization layer.

over each dimension of the 3D data volume. As a result, **S2** contains a total of six operators, of which four are weight-equipped convolutional parameters, and the average number of operator parameters are reduced by 65%.

5 Experiments and Results

In this chapter, we provide a detailed explanation of the experimental procedure of our approach. Further, we address the following research questions: 1) whether PC-DARTS discovers fusion architectures that achieve superior performance than the simple feature concatenation strategy, and 2) how the macro-architecture design settings affect the performance and the model size of the resulting fusion architecture. We answer these questions through the obtained experimental results and its analysis.

5.1 Dataset

5.1.1 HMDB51

HMDB51 [Kuehne et al., 2011] consists of 51 action classes with a total of 6,766 video clips extracted from a wide range of sources. It is one of the more challenging datasets because the dataset size is relatively small, it contains lots of actions that are similar to each other, and the samples are collected from a wide range of sources. The dataset has three different split settings of training and test samples, where the training and test set contain 5,100 and 1,666 samples, respectively. Our experiments are mainly based on HMDB51, and we report our experimental results on the split 1.

5.1.2 UCF101

Further, we conduct several experiments on UCF101 [Soomro et al., 2012], which is another popular dataset for benchmarking the performance of video action classification models. It contains 13,320 video clips of 101 classes collected from various web sources. Like HMDB51, it has three splits of train/test sets containing 9,537 and 3,783 clips, respectively. We also use split 1 to report the test performance of our approach.

5.1.3 Frame sampling

Wang et al. [2016] showed that using all or densely sampled frames of a video clip is inefficient since information contained in consecutive frames are highly redundant, and that sparsely sampled frames can maintain the global temporal structure in the data but are significantly more computationally efficient due to the smaller size of the data.

Accordingly, we employ a sparse frame sampling scheme for selecting the frames in video clips to use for the classification task. The sampling procedure is as follows. First, we rescale the height and width of the video clip V to have the input volume dimension of $[T, C, 112, 112]$, where T and C are the frame length and the number of channels (3 for RGB and pose, 2 for optical flow), respectively. Then, we split V into 64 segments $\{S_1, S_2, ..., S_{64}\}$ of equal lengths, such that the dimension of each segment is $[\frac{T}{64}, C, 112, 112]$. Then, for the training set, we randomly sample a frame from each segment and combine them to form a new clip V_{new} that has 64 frames. For the validation and test set, we randomly select a position in a segment and sample the frame corresponding to that position from all segments, such that all sampled frames are equidistant.

Formally, let $f(S, i)$ be a function that outputs the frame corresponding to the i-th position in the segment S of length T. Then, the new clip after the sampling can be written as:

$$\begin{aligned} V_{train} &= concat(f(S_1, i_1), f(S_2, i_2), ..., f(S_{64}, i_{64})) \\ V_{val,test} &= concat(f(S_1, p), f(S_2, p), ..., f(S_{64}, p)) \end{aligned} \tag{9}$$

where $\{i_1, ..., i_{64}\}$ and p are sampled from a discrete uniform distribution $\mathcal{U}(1, T)$.

This segment-based sampling scheme converts the video data of various durations to clips of equal lengths, enabling the training of our model. Further, randomly sampling a frame from each segment for the training set has the effect of data augmentation since multiple combinations of frames can be generated from a single dataset.

Although rare, it is possible that a video clip contains less frames than the number of segments. In that case, we repeat the frame sequence from the beginning multiple times until it has 64 frames such that it can be segmented. Also, if the number of frames in a clip is not an exact multiple of 64, which is almost always the case, we truncate the minimal number of frames at the beginning and the end of the clip to make it an exact multiple of 64. Although this may result in some loss of information,

in practice it does not hurt the performance since the sequences in the middle of the video where the action usually occurs is kept.

5.2 Experimental Design

5.2.1 Architecture Settings

As discussed in the approach section 4.2, we form a 4-dimensional search space where each axis represent the number of cell input connection $I \in \{1, 2, 3\}$, number of stacked cells $C \in \{1, 2, 3, 4\}$, cell discretization method $disc \in \{all, topk\}$, and the set of candidate operators $S \in \{\mathbf{S1}, \mathbf{S2}\}$, respectively, and perform a grid search by running each possible configuration of $(I, C, disc, S)$ six times to obtain the best performance.

5.2.2 Implementation Detail

For each run, we first obtain the architecture by performing PC-DARTS for 50 epochs. We split the training set into two subsets of equal sizes and optimize the model parameters w on the training loss, and the architecture parameters α on the validation loss, using the first and the second subsets respectively. The hyperparamter K controlling the proportion of sampled channels in PC-DARTS is set to 4, i.e., 25% of the channels are sampled at each node of the cells. For training the search architecture, we set the batch size to 64 and the initial learning rates of w and α to $2e-3$ and $3e-4$, respectively. We use stochastic gradient descent with momentum of 0.9, and cosine annealing [Loshchilov and Hutter, 2016] without warmstart to gradually decrease the architecture learning rate to 0, in order to help the search architecture converge over the epochs. Moreover, to inhibit the parameter-less operators ($SkipConnect$, $zero$) from dominating at earlier stages of the architecture search, we warm up the architecture search by freezing α and only training w in the first 5 epochs.

During the search, the current snapshot of the architecture and its validation loss is stored at the end of each epoch. Afterwards, the fusion architecture is derived by selecting the architecture snapshot with the best validation loss and applying discretization to it. The resulting network is then trained from scratch for 70 epochs. At this stage, we resplit the training set randomly into two subsets of ratio 90/10, and the network is trained on the larger subset while the smaller subset is used to compute the validation loss at the end of each epoch. Finally, we report the test set performance of the network snapshot with the best validation loss. The hyperparameters used for

evaluating the network is the same as those used for searching the network.

5.2.3 Baseline

To measure the improvement made by the fusion architectures, we compare our experimental results with those of the baseline architectures, illustrated in figure 8. Our baseline architectures consist of the pre-trained modality streams but does not fuse the features; instead, it simply concatenates the modality features channel-wise, applies global average pooling to eliminate the spatio-temporal resolution of the features, and feed them to the fully connected layer to create the final prediction. We note that the baseline architectures use the outputs of the fourth layer of the modality streams instead of the third layer, since it does not contain the fuse cells and therefore the spatio-temporal resolution of the stream outputs no longer need to be sufficiently large.

Further, to study the contribution of each stream to the overall performance, we construct seven versions of the baseline architecture, each containing a possible combination of one or more streams as follows: $[R,\ F,\ P,\ R+F,\ R+P,\ F+P,\ R+F+P]$, where R, F, P denote RGB, flow and pose, respectively.

The training procedure of the baselines is same as that of the fusion architecture: the weights of the fully connected layer is trained for 70 epochs, and the weights of the pre-trained models are frozen. For optimization, we use SGD with momentum of 0.9 and use cosine annealing to decrease initial learning rate of $2e-3$ over the epochs, and add $L2$ regularization on the parameters with a factor of $3e-4$. The snapshot of the architecture is stored at the end of each epoch, and the test accuracy of the snapshot with the best validation error is reported. Each baseline run is repeated six times to obtain the best result.

5.3 Results

5.3.1 Results on HMDB51

Baseline

Table 3 shows the best results of each of our architecture settings as well as the seven baseline architectures on HMDB51. The baseline experiments show that while all three modality streams positively contribute to the increase in performance, the contribution of the RGB and flow streams are more significant than that of the pose stream. This is presumably due to the fact the stream used for pose was pre-trained

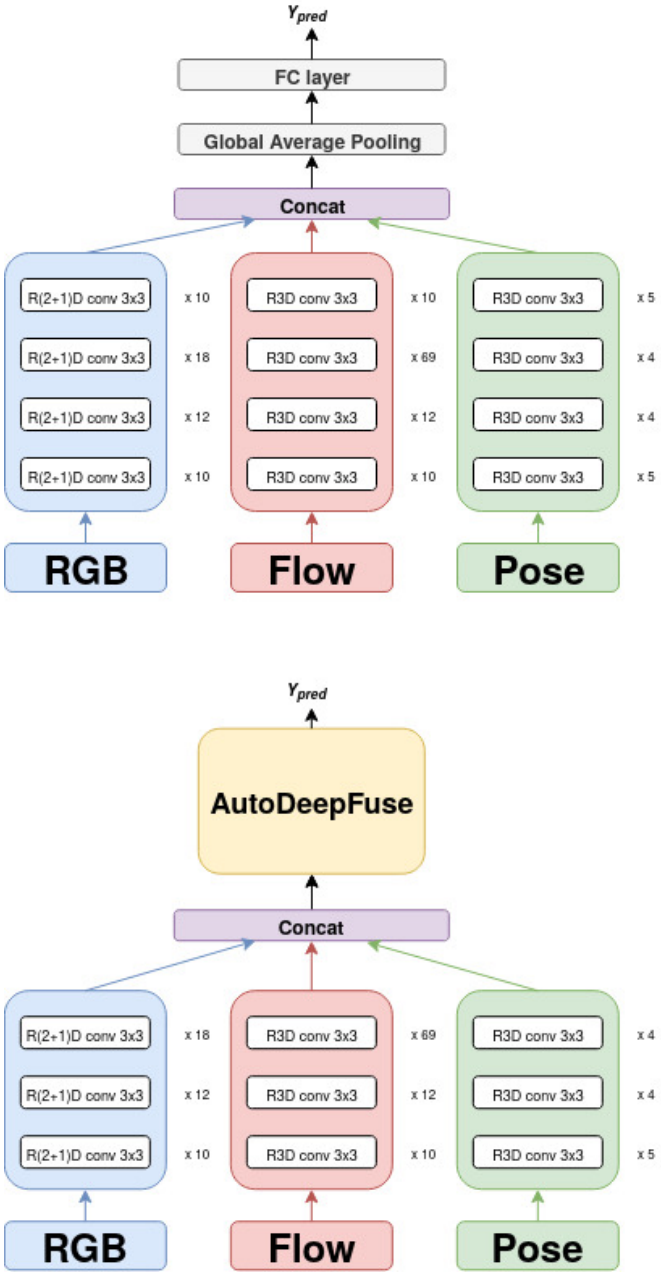

Figure 8: the fusion architecture (bottom) and the baseline (top) that uses all three modality streams. The fusion architecture fuses the outputs of the third layer of modality streams, while the baseline takes the outputs of the fourth layer of the streams and applies global average pooling and a fully connected layer.

Baseline	Top-1 acc.	# Param. (M)
R	60.69	46
F	61.35	47
P	24.75	33
R+F	68.87	93
R+P	62.50	79
F+P	62.37	80
R+F+P	**68.93**	127

Table 3: Test performance of the baseline architectures. The performance increases when more modalities are fused.

	PC-DARTS			Random		
Architecture Setting	# Cells	Top-1 acc.	# Param. (M)	# Cells	Top-1 acc.	# Param. (M)
S1, all, I = 1	3	73.97	236	2	73.81	228
S1, all, I = 2	3	**75.24**	274	2	74.81	259
S1, all, I = 3	1	73.91	222	3	**74.45**	312
S1, top-k, I = 1	2	74.87	115	4	**74.91**	162
S1, top-k, I = 2	4	**75**	179	2	74.89	128
S1, top-k, I = 3	2	74.63	178	3	72.47	189
S2, all, I = 1	2	74.45	103	2	74.39	95
S2, all, I = 2	2	**75.72**	117	3	**75.39**	124
S2, all, I = 3	2	75.12	124	2	73.78	117
S2, top-k, I = 1	2	**75.48**	88	3	73.12	103
S2, top-k, I = 2	2	75.24	95	2	**74.74**	90
S2, top-k, I = 3	2	74.51	109	2	74.67	99
S2, all, I = 2 (BOHB)	2	**76.21**	117			

Table 4: Best test performance of each architecture setting (left) and the randomly sampled architectures (right).

on RGB. Nevertheless, the results show that combining the pose stream always yields better performance than when it is excluded. The baseline performances of RGB + pose, flow + pose, and RGB + flow + pose are higher than the performances of RGB, flow, and RGB + flow by 1.81%, 1.02% and 0.06%, respectively. As expected, the best performance was obtained when all three modalities were fused. The test accuracy of the all-stream model reached 68.93%.

Search & Random Architectures

Table 4 shows the test performance of the best discovered architecture of each architecture setting $(S, disc, I)$. Results show that all fusion architectures searched via PC-DARTS significantly outperformed the baseline. The architecture setting $(\mathbf{S2}, all, I = 2)$ discovered the best architecture, reaching the highest accuracy at 75.72%, which is 6.79% higher than the baseline, while the worst setting was

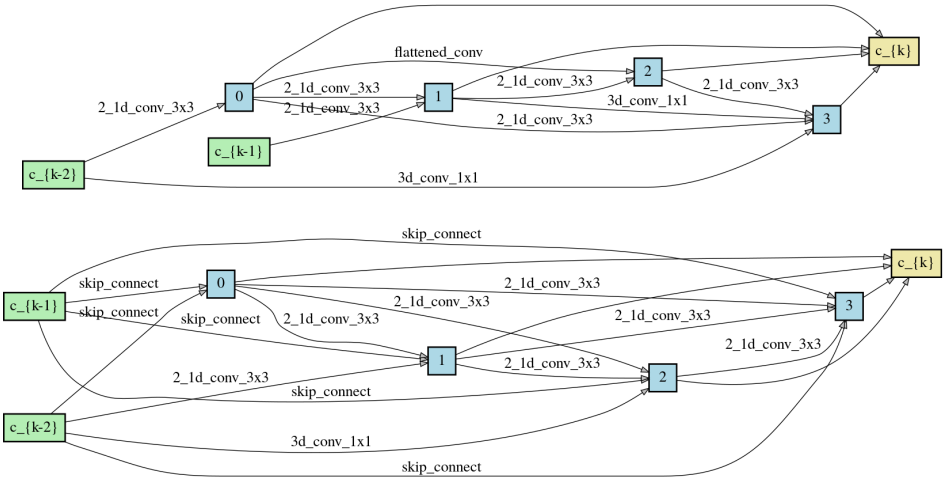

Figure 9: The best architecture (75.72%) discovered through PC-DARTS Cell 1 and Cell 2 are illustrated at the top and the bottom of the figure, respectively. When the hyperparameters are optimized, the accuracy was improved to 76.21%.

($\mathbf{S1}, all, I = 3$) with the accuracy at 73.91%. The best found architecture is illustrated in figure 9. Also, to verify that PC-DARTS performs a proper architecture search rather than randomly selecting operators, we compare the performance of the found architectures with the randomly sampled architectures. For each setting, a random architecture is constructed by randomly sampling operators from the candidate set and evaluated with the same procedure as the searched architectures. Results show that the searched architectures generally performed better than the random architectures. However, for the settings ($S1, all, I = 3$), ($S1, topk, I = 1$) and ($S2, topk, I = 3$), the random architectures performed slightly better than the searched architectures, suggesting that effective architectures can be sampled by chance from the given search space.

We also optimized hyperparameters of the best found architecture via BOHB [Falkner et al., 2018], and achieved further improvement in performance at 76.21%. The optimized hyperparameters and details of the BOHB run is explained in table 5.

The model size of the discovered architectures varied widely depending on their architecture settings, but no clear correlation was observed between the model size and the performance. In fact, all architectures discovered using the reduced operator set **S2** achieved slightly higher accuracy compared to their **S1** counterparts with only about 50% of the model size. The two best models found using **S2**, that achieved

Hyperparameter	Type	Range/Choice	Log scale
batch size	categorical	[4, 8, 16, 32]	
learning rate	float	[1e-5, 1e-2]	yes
optimizer	categorical	[Adam, Adamw, SGD]	
L2 factor	float	[1e-5, 2e-2]	yes

Table 5: BOHB hyperparameter search space. BOHB was run for 10 iterations with the mininum and maximum budgets of 2 and 70 epochs, respectively, and $\eta = 3$

the accuracy of 75.72% and 75.48% respectively, contained 10 million and 29 million parameters less than the baseline, while the two best architectures discovered using **S1** contained 157 million and 62 million parameters more but performed worse than those found using **S2**. Moreover, regarding the number of stacked cells C and cell connectivity I, we detected a tendency of performance decrease when more parameters were used. In general, an architecture achieved high performance when $C = 2$ and $I = 2$, where the model size was neither too small to cause underfitting nor too large that it is either too hard to train or over-fits to the data.

Our experiments clearly suggest that, while PC-DARTS is able to discover efficient and well-performing modality fusion architectures, the macro-level architecture design choices are pivotal for fully leveraging the power of PC-DARTS. Therefore, we further analyze the effect of each design choices on the performance in the sections below.

Effect of Cell Stacking

First, we analyze the effect of cell stacking on the resulting performance. According to table 4, the architecture search generally discovers the best fusion networks when two cells are stacked. Out of 12 architecture settings, 8 of them produced the best result when $C = 2$. In particular, stacking two cells always performed best when the small candidate operator set **S2** is used. We also observe this tendency when the results of each C are averaged. Figure 10 illustrates the effect of the number of stacked fuse cells, for each candidate operator set used, on the resulting test accuracy, where each box represents the performance distribution of each C across all other settings. We observe that the average accuracy is the highest when two cells were stacked, while the accuracy gradually decreases when more cells are stacked to build deeper architectures. On the other hand, stacking only one cell to build shallow networks resulted in under-fitting to the data. This phenomenon was especially prominent when **S2** was used. The results indicate that simply stacking more cells does not

necessarily result in a better network.

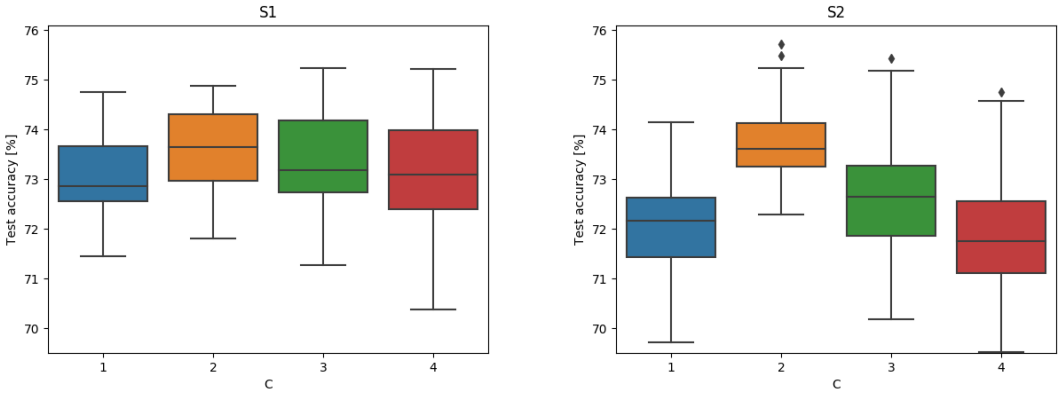

Figure 10: Effect of number of stacked cells on test accuracy with **S1** (left) and **S2** (right). In both cases, $C = 2$ yields the best average performance, and the performance gradually decreases as more cells are stacked.

Effect of Candidate Operator Set

Next, we analyze the effect of candidate operators set used for the architecture search. Results indicate that PC-DARTS tends to either select operators with the most number of parameters in the operator set, or *SkipConnect*. As illustrated in figure 11, in **S1**, $3 \times 3 \times 3$ *Conv*, $3 \times 3 \times 3$ *DilConv* and *SkipConnect* are the most selected operators, while other operators are hardly chosen. Similarly, in **S2**, $(2 + 1)D\_Conv$ and *SkipConnect* are the most frequent operators. The phenomenon of parameter-less operators such as *SkipConnect* dominating the optimized architecture is a well-known problem, and the results show that a similar tendency, although not as extreme, exists in our experiments. On the other hand, while the fact that operators with larger parameter size is preferred may suggest that they are the best choices for the fusion task, the performance of the derived architectures do not necessarily correlate with the ratio of these operators. When **S1** is used, PC-DARTS hardly selected $(2 + 1)D\_Conv$, suggesting that it is a suboptimal operator for our task, However, when **S2** is used, $(2 + 1)D\_Conv$ is heavily preferred over other parameters, and the discovered architectures with a high ratio of $(2 + 1)D\_Conv$ tended to perform better than the architectures found using **S1**. This finding indicates that operators deemed optimal by DARTS in the search phase are not always the best operators in the discretized architecture, and therefore a careful design of the operator set by the DARTS implementer is still neccesary to obtain good performance.

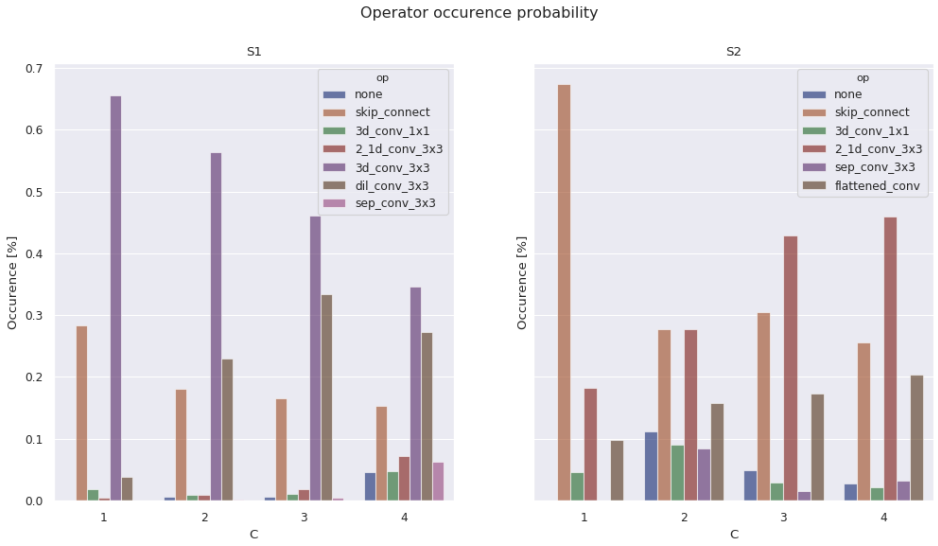

Figure 11: Bar charts of the probability PC-DARTS selecting each operator. Both candidate operator sets **S1** and **S2** are shown.

Effect of Cell Connectivity

Figure 12 illustrates the performance of each combination of (I, $disc$) averaged across all experiments. As shown in the figure, PC-DARTS discovers better architectures when $I = 2$ is used. Because I controls the size of the architecture, we observe a phenomenon similar to that we see in the analysis of C. When $I = 1$, the size of the discovered architectures contain relatively few parameters. Therefore, although they are more efficient in terms of computational resources, their performance is worse than when $I = 2$. On the contrary, when $I = 3$, the networks are large and the connections between the cells are dense, thus they are harder to train.

Effect of Discretization Method

Regarding the discretization method, although the *all* method performs slighlty better than the *topk* method, we do not observe a significant performance difference. However, since the *topk* method prunes a large proportion of the node connections, the overall number of parameters retained in architectures derived using the *topk* are significantly less than those obtained using the *all* method. Hence, results suggest that, to find an efficient fusion network, discretizing the search architecture using the *topk* method is a reasonable choice without much loss in performance.

However, we also observe that the variance of the run results obtained with *topk*

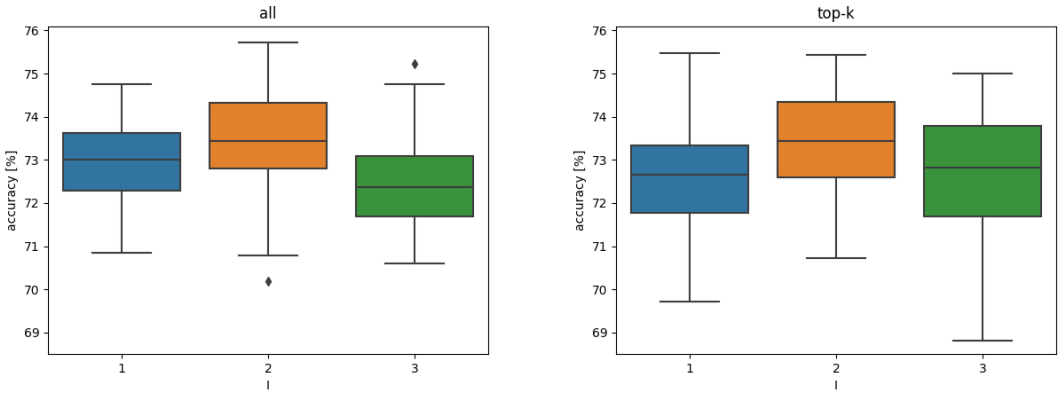

Figure 12: Effect of cell connectivity on the test accuracy with *all* (left) and *topk* (right). $I = 2$ produces the best result in both cases.

methods are higher than the results of the *all* method, as illustrated in figure 13. As we have hypothesized, it is likely that arbitrarily pruning optimized connections inside the cells introduced a larger optimization gap, rendering the search architecture's performance estimate of the derived architecture less accurate. The results indicate that, although the performance variance caused by this gap is not as extreme in our experiments, this phenomenon may be more troublesome in other application of DARTS, and therefore the search space design must be meticulously done in order to avoid suboptimal performance.

5.3.2 Transfer to UCF101

We also applied transfer learning by evaluating the networks discovered with HMDB51 on UCF101. Table 6 shows the transfer learning results on UCF101 as well as the baseline performance. Results show that the networks found with HMDB51 also achieve good performance when transferred to the larger dataset. All of the networks yielded higher performance on UCF101 than the best baseline architecture that achieves the accuracy of 92.78%.

5.3.3 Comparison with Other Approaches

Table 7 shows the performance of our and other approaches on HMDB51 and UCF101. Comparing with other NAS methods, AutoDeepFuse does not perform better than EvaNet [Piergiovanni et al., 2019] (on HMDB51). However, we believe that the runtime of our architecture search method is orders of magnitude faster than that of

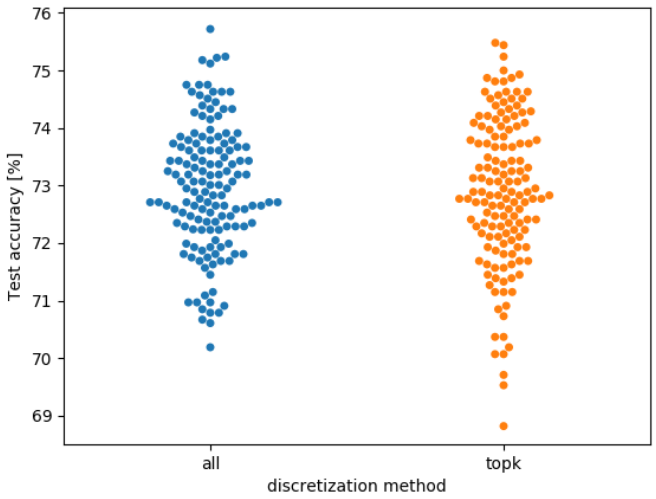

Figure 13: Swarm plot of experimental results obtained via each discretization method. $Var(X_{all}) = 1.25$, $Var(X_{topk}) = 1.69$.

EvaNet. Although EvaNet's search runtime is not explicitly stated in their paper, their method adopts an evolutionary strategy in which thousands of candidate architectures are completely evaluated. On the other hand, our method requires merely 2 GPU days for the complete search and evaluation.

Our optimal fusion network, when transferred to UCF101, performs significantly better (on UCF101) than the simple DARTS approach [Peng et al., 2019] that directly uses the raw RGB data of UCF101 for the search. The runtime of their architecture search (excluding evaluation) is 25 GPU hours, showing that our method improves the accuracy by more than 35% with only slight increase in the runtime.

Comparing with the manual approaches, AutoDeepFuse outperforms the two-stream networks with simple fusion schemes, but does not outperform the state-of-the-art architectures such as I3D and MARS. This is presumably due to the fact that, while all of these methods fine-tune their architectures after pre-training, we do not fine-tune the parameters of the pre-trained models, which take up most of the parameters in the architecture. Therefore, the final performance of our model is bound by the original performance of the used pre-trained models. However, since our experiments suggest that the network is able to significantly increase the performance of the pre-trained models via effective fusion, we believe that our approach has a potential to yield superior performance when state-of-the-art pre-trained backbones are used.

UCF101	Transfer Learning		
Architecture Setting	# Cells	Top-1 acc.	# param. (M)
S1, all, I = 1	3	94.06	236
S1, all, I = 2	3	**94.35**	274
S1, all, I = 3	1	94.31	222
S1, top-k, I = 1	2	93.56	115
S1, top-k, I = 2	4	93.77	179
S1, top-k, I = 3	2	**94.14**	178
S2, all, I = 1	2	93.76	103
S2, all, I = 2	2	**94.22**	117
S2, all, I = 3	2	94.13	124
S2, top-k, I = 1	2	93.59	88
S2, top-k, I = 2	2	**94.02**	95
S2, top-k, I = 3	2	93.95	109
Baseline			
R		89.92	46
F		85.61	47
P		32.65	33
R+F		92.30	93
R+P		90.51	79
F+P		86.71	80
R+F+P		**92.78**	127

Table 6: UCF101 test accuracy of best architectures found with HMDB51 and baseline architectures.

Method	HMDB51	UCF101
Manual		
Two-Stream [Simonyan and Zisserman, 2014]	59.4	88.0
Two-Stream Fusion [Feichtenhofer et al., 2016]	65.4	92.5
C3D [Tran et al., 2015]		85.2
R(2+1)D [Tran et al., 2018]	74.5	
I3D [Carreira and Zisserman, 2017]	80.1	97.8
MARS [Crasto et al., 2019]	80.9	95.8
NAS		
EvaNet [Piergiovanni et al., 2019]	82.3	
DARTS [Peng et al., 2019]		58.6
AutoDeepFuse	75.72	94.35
AutoDeepFuse (BOHB)	76.2	

Table 7: Comparison of top-1 test accuracy (in percentage) with other approaches. EvaNet performs better (on HMDB51) than our approach, but at the cost of runtime orders of magnitude larger than AutoDeepFuse, since it requires complete evaluation of thousands of candidate architectures. On the other hand, AutoDeepFuse performs significantly better (on UCF101) than the simple DARTS approach with the increase in the runtime of less than one GPU day.

6 Conclusion

In this paper, we presented a novel approach to the task of video action recognition via multi-modal stream fusion, in which we employed a 3D spatio-temporal CNN to efficiently fuse the features of the RGB, optical flow, and pose estimate of the data extracted from the pre-trained modality streams. Instead of manually designing the fusion scheme, we automatically discovered optimal fusion networks via PC-DARTS, an efficient neural architecture search method, and showed that the found networks are able to increase the performance of the pre-trained streams through effective fusion of the modality features.

Moreover, we showed that while DARTS is able to find well-performing architectures, their quality is also subject to the macro-level design decisions left to the implementer. Through a set of systematic experiments, we found that these design choices have a heavy impact the performance and the size of the discovered architectures. Specifically, our experiments indicated that, when a large search space is given, DARTS tends to find networks with a large model size, but the network, when discretized, are not necessarily optimal. Rather, we observed a tendency of smaller (but not too small to cause under-fitting) networks to perform better. Also, our finding supports the belief that discretization methods that heavily distorts the search architecture may fail to adequately reflect the true performance of the derived architecture and can lead to higher variance in evaluation.

For further improving the performance of our approach, future research directions include experimentation with additional input modalities, such as object segmentation, with different pre-trained modality streams, and with other types of 3D spatio-temporal operators.

7 Acknowledgments

I would like to express my gratitude to my supervisors Arber Zela and Mohammadreza Zolfaghari for the constructive discussions, valuable feedbacks and academic and personal support throughout the research and my learning process, without which this thesis would not have been possible. Further, I would like to thank Professor Frank Hutter for providing a great research environment and comfortable, friendly laboratory atmoshpere. I would also like to thank Professor Thomas Brox for examining my thesis and providing insightful comments for further improvements. Moreover, I would like to thank all of my friends and family who supported me in various ways for making my master's study in Freiburg a great experience by keeping me motivated and spending great times together.

Bibliography

Mykhaylo Andriluka, Leonid Pishchulin, Peter Gehler, and Bernt Schiele. 2d human pose estimation: New benchmark and state of the art analysis. In *Proceedings of the IEEE Conference on computer Vision and Pattern Recognition*, pages 3686–3693, 2014.

Svitlana Antoshchuk, Mykyta Kovalenko, and Jürgen Sieck. Gesture recognition-based human–computer interaction interface for multimedia applications. In *Digitisation of Culture: Namibian and International Perspectives*, pages 269–286. Springer, 2018.

Sina Mokhtarzadeh Azar, Mina Ghadimi Atigh, and Ahmad Nickabadi. A multi-stream convolutional neural network framework for group activity recognition. *arXiv preprint arXiv:1812.10328*, 2018.

Mohamed Ben Youssef, Imen Trabelsi, and Med Bouhlel. Human action analysis for assistance with daily activities. *International Journal on Human Machine Interaction*, 07 2016.

Victoria Bloom. *Multiple Action Recognition for Video Games (MARViG)*. PhD thesis, Kingston University, 2015.

Andrew Brock, Theodore Lim, James M Ritchie, and Nick Weston. Smash: one-shot model architecture search through hypernetworks. *arXiv preprint arXiv:1708.05344*, 2017.

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.

Xin Chen, Lingxi Xie, Jun Wu, and Qi Tian. Progressive darts: Bridging the optimization gap for nas in the wild. *International Journal of Computer Vision*, pages 1–18, 2020.

Ronan Collobert, Jason Weston, Léon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel Kuksa. Natural language processing (almost) from scratch. *Journal of machine learning research*, 12(ARTICLE):2493–2537, 2011.

Nieves Crasto, Philippe Weinzaepfel, Karteek Alahari, and Cordelia Schmid. Mars: Motion-augmented rgb stream for action recognition. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 7882–7891, 2019.

Jeffrey Donahue, Lisa Anne Hendricks, Sergio Guadarrama, Marcus Rohrbach, Subhashini Venugopalan, Kate Saenko, and Trevor Darrell. Long-term recurrent convolutional networks for visual recognition and description. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2625–2634, 2015.

Xuanyi Dong and Yi Yang. Searching for a robust neural architecture in four gpu hours. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1761–1770, 2019.

Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. Neural architecture search: A survey. *arXiv preprint arXiv:1808.05377*, 2018.

Stefan Falkner, Aaron Klein, and Frank Hutter. Bohb: Robust and efficient hyperparameter optimization at scale. In *International Conference on Machine Learning*, pages 1437–1446. PMLR, 2018.

Christoph Feichtenhofer, Axel Pinz, and Andrew Zisserman. Convolutional two-stream network fusion for video action recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1933–1941, 2016.

Christoph Feichtenhofer, Axel Pinz, and Richard P. Wildes. Spatiotemporal multiplier networks for video action recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017.

Malik Ali Gul, Muhammad Haroon Yousaf, Shah Nawaz, Zaka Ur Rehman, and HyungWon Kim. Patient monitoring by abnormal human activity recognition based on cnn architecture. *Electronics*, 9(12):1993, 2020.

Kensho Hara, Hirokatsu Kataoka, and Yutaka Satoh. Learning spatio-temporal features with 3d residual networks for action recognition. In *Proceedings of the*

IEEE International Conference on Computer Vision Workshops, pages 3154–3160, 2017.

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

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

Hueihan Jhuang, Juergen Gall, Silvia Zuffi, Cordelia Schmid, and Michael J Black. Towards understanding action recognition. In *Proceedings of the IEEE international conference on computer vision*, pages 3192–3199, 2013.

Jonghoon Jin, Aysegul Dundar, and Eugenio Culurciello. Flattened convolutional neural networks for feedforward acceleration. *arXiv preprint arXiv:1412.5474*, 2014.

Kirthevasan Kandasamy, Willie Neiswanger, Jeff Schneider, Barnabas Poczos, and Eric Xing. Neural architecture search with bayesian optimisation and optimal transport. *arXiv preprint arXiv:1802.07191*, 2018.

Andrej Karpathy, George Toderici, Sanketh Shetty, Thomas Leung, Rahul Sukthankar, and Li Fei-Fei. Large-scale video classification with convolutional neural networks. pages 1725–1732, 2014.

Hirokatsu Kataoka, Tenga Wakamiya, Kensho Hara, and Yutaka Satoh. Would mega-scale datasets further enhance spatiotemporal 3d cnns?, 2020.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25:1097–1105, 2012.

Hildegard Kuehne, Hueihan Jhuang, Estíbaliz Garrote, Tomaso Poggio, and Thomas Serre. Hmdb: a large video database for human motion recognition. In *2011 International Conference on Computer Vision*, pages 2556–2563. IEEE, 2011.

Liam Li and Ameet Talwalkar. Random search and reproducibility for neural architecture search. In *Uncertainty in Artificial Intelligence*, pages 367–377. PMLR, 2020.

Lisha Li, Kevin Jamieson, Giulia DeSalvo, Afshin Rostamizadeh, and Ameet Talwalkar. Hyperband: A novel bandit-based approach to hyperparameter optimization. *The Journal of Machine Learning Research*, 18(1):6765–6816, 2017.

Chenxi Liu, Barret Zoph, Maxim Neumann, Jonathon Shlens, Wei Hua, Li-Jia Li, Li Fei-Fei, Alan Yuille, Jonathan Huang, and Kevin Murphy. Progressive neural architecture search. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 19–34, 2018a.

Hanxiao Liu, Karen Simonyan, and Yiming Yang. Darts: Differentiable architecture search. *arXiv preprint arXiv:1806.09055*, 2018b.

Ilya Loshchilov and Frank Hutter. Sgdr: Stochastic gradient descent with warm restarts. *arXiv preprint arXiv:1608.03983*, 2016.

Gabriel L Oliveira, Wolfram Burgard, and Thomas Brox. Efficient deep models for monocular road segmentation. In *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4885–4891. IEEE, 2016.

Eunbyung Park, Xufeng Han, Tamara L Berg, and Alexander C Berg. Combining multiple sources of knowledge in deep cnns for action recognition. In *2016 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 1–8. IEEE, 2016.

Wei Peng, Xiaopeng Hong, and Guoying Zhao. Video action recognition via neural architecture searching. In *2019 IEEE International Conference on Image Processing (ICIP)*, pages 11–15. IEEE, 2019.

Hieu Pham, Melody Guan, Barret Zoph, Quoc Le, and Jeff Dean. Efficient neural architecture search via parameters sharing. In *International Conference on Machine Learning*, pages 4095–4104. PMLR, 2018.

AJ Piergiovanni, Anelia Angelova, Alexander Toshev, and Michael S Ryoo. Evolving space-time neural architectures for videos. In *Proceedings of the IEEE international conference on computer vision*, pages 1793–1802, 2019.

Esteban Real, Sherry Moore, Andrew Selle, Saurabh Saxena, Yutaka Leon Suematsu, Jie Tan, Quoc Le, and Alex Kurakin. Large-scale evolution of image classifiers. *arXiv preprint arXiv:1703.01041*, 2017.

Alina Roitberg, Tim Pollert, Monica Haurilet, Manuel Martin, and Rainer Stiefelhagen. Analysis of deep fusion strategies for multi-modal gesture recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 0–0, 2019.

Binxin Ru, Xingchen Wan, Xiaowen Dong, and Michael Osborne. Neural architecture search using bayesian optimisation with weisfeiler-lehman kernel. *arXiv preprint arXiv:2006.07556*, 2020.

Michael S Ryoo, AJ Piergiovanni, Mingxing Tan, and Anelia Angelova. Assemblenet: Searching for multi-stream neural connectivity in video architectures. *arXiv preprint arXiv:1905.13209*, 2019.

Bobak Shahriari, Kevin Swersky, Ziyu Wang, Ryan P Adams, and Nando De Freitas. Taking the human out of the loop: A review of bayesian optimization. *Proceedings of the IEEE*, 104(1):148–175, 2015.

Karen Simonyan and Andrew Zisserman. Two-stream convolutional networks for action recognition in videos. In *Advances in neural information processing systems*, pages 568–576, 2014.

Khurram Soomro, Amir Roshan Zamir, and Mubarak Shah. Ucf101: A dataset of 101 human actions classes from videos in the wild. *arXiv preprint arXiv:1212.0402*, 2012.

Lin Sun, Kui Jia, Dit-Yan Yeung, and Bertram E Shi. Human action recognition using factorized spatio-temporal convolutional networks. In *Proceedings of the IEEE international conference on computer vision*, pages 4597–4605, 2015.

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.

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.

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.

Limin Wang, Yuanjun Xiong, Zhe Wang, Yu Qiao, Dahua Lin, Xiaoou Tang, and Luc Van Gool. Temporal segment networks: Towards good practices for deep action recognition. In *European conference on computer vision*, pages 20–36. Springer, 2016.

Saining Xie, Ross Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He. Aggregated residual transformations for deep neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1492–1500, 2017.

Yuhui Xu, Lingxi Xie, Xiaopeng Zhang, Xin Chen, Guo-Jun Qi, Qi Tian, and Hongkai Xiong. Pc-darts: Partial channel connections for memory-efficient differentiable architecture search. *arXiv preprint arXiv:1907.05737*, 2019.

Kaicheng Yu, Christian Sciuto, Martin Jaggi, Claudiu Musat, and Mathieu Salzmann. Evaluating the search phase of neural architecture search. *arXiv preprint arXiv:1902.08142*, 2019.

Joe Yue-Hei Ng, Matthew Hausknecht, Sudheendra Vijayanarasimhan, Oriol Vinyals, Rajat Monga, and George Toderici. Beyond short snippets: Deep networks for video classification. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4694–4702, 2015.

Christopher Zach, Thomas Pock, and Horst Bischof. A duality based approach for realtime tv-l 1 optical flow. In *Joint pattern recognition symposium*, pages 214–223. Springer, 2007.

Arber Zela, Aaron Klein, Stefan Falkner, and Frank Hutter. Towards automated deep learning: Efficient joint neural architecture and hyperparameter search. *arXiv preprint arXiv:1807.06906*, 2018.

Arber Zela, Thomas Elsken, Tonmoy Saikia, Yassine Marrakchi, Thomas Brox, and Frank Hutter. Understanding and robustifying differentiable architecture search. *arXiv preprint arXiv:1909.09656*, 2019.

Mohammadreza Zolfaghari, Gabriel L Oliveira, Nima Sedaghat, and Thomas Brox. Chained multi-stream networks exploiting pose, motion, and appearance for action classification and detection. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2904–2913, 2017.

Barret Zoph and Quoc V Le. Neural architecture search with reinforcement learning. *arXiv preprint arXiv:1611.01578*, 2016.