A3P: Audio as a Preview for Efficient Activity Recognition in Video

by

Andrew Li

Supervised by:
Dr. Kristen Grauman

Department of Computer Science
The University of Texas at Austin

Acknowledgements

I would like to thank my advisor Dr. Kristen Grauman for providing guidance and direction throughout this project, and Tushar Nagarajan for helping to resolve difficulties encountered during the writing process and for providing a significant amount of technical assistance during the project itself. My project would not be where it is now without the help of either of them.

Abstract

Action recognition has become an increasingly popular task, but most modern methods based on deep convolutional neural networks fail to consider the computational cost of the models. When working with frames, there is a trade-off between selecting a dense set of frames, which covers the entire video but have a high computational cost, and a sparse set of frames, which have a low computational cost but may skip over important frames. In contrast, with audio one can get the best of both worlds: one can densely summarize a video with a number of parameters comparable to that of only a few frames. We propose Audio as a Preview (A3P), which treats audio data as a summary of a video clip to intelligently select which frames of a video are processed for visual features when performing activity recognition. Our system attempts to both choose frames which are conducive to classification, as well as minimize the number of frames actually selected. We evaluate our model on two challenging first-person video datasets: GTEA Gaze+ and EPIC-KITCHENS, and experiment with several variants of our model. Our method performs better than several baselines we considered, including naive implementations of audio-based frame selection, though certain other baselines outperform it. Based on our results, we propose further improvements which could be made.

Contents

List of Figures

List of Tables

1 Introduction

The problem of action recognition appears in a wide variety of settings. For example, action recognition algorithms can be used to analyze trends in websites which allow uploading of user-generated videos. They can be used to determine what action a human is performing for a robot, or analyzing the behaviors of people in public. Of course, action recognition is not limited to humans; such algorithms can also be applied to, for example, videos of nature or industrial processes.

There have been numerous approaches to this task [11, 14, 23], but most of them suffer from a significant problem: that of computation power. Specifically, many modern deep-learning based models have to process a dense set of frames in order to make a classification. This requires a very large amount of computational resources, which is often impractical. For example, when running an action classification model on embedded devices (such as mobile phones), the processor might not run fast enough to process the video in real-time. Even in a traditional batch setting where all videos are provided at once, the high computational requirements could make the difference between an action recognition task completing in one day versus completing in months.

Intuitively, one may feel that this dense sampling of frames is not necessary. A human, if asked to classify a long video, would probably not attempt to watch the entire video before finally making a prediction. Instead, the human would skip around the video, spending very little time where there is "obviously" nothing useful occurring, and looking in more detail when something interesting occurs. Existing works have in fact proposed systems to perform such an iterative scanning process [24, 38, 7, 36]. However, these methods all share some disadvantages. One disadvantage is that frames are processed in sequence; there is no way to start processing the next frame (or even know what it is!) before most of the computation for the current step has completed. More notably, they all ignore a significant modality: audio. In many applications of action recognition and video datasets, audio data is present but not used. However, audio is very cheap to manipulate compared to a large block of frames, and it can provide temporally localized information about actions occurring at particular times.

To address this issue, we introduce Audio as a Preview (A3P), where we treat the audio data as a preview of the entire video clip. Specifically, our model learns to select, based on the audio data of the video clip, the visual frames of the video clip which are most useful for classification. The model then looks at only those frames which were chosen, before producing the final classification. This idea is illustrated in Figure 1. We also design the frame-level classifier so that each frame is independently processed until the last few layers, which in theory allows for parallelizing the expensive early part of the computation.

We evaluate our method by running on two datasets: GTEA Gaze+ [8] and EPIC-KITCHENS [4]. We also experiment with variants of our method that change how parts of our model are trained. We find that our proposed method, run as is, has some promise for choosing a selection based on the audio. Finally, we discuss factors which may have impacted these results, and propose improvements for further study.

2 Related Work

We first present some related work on action recognition and adaptive computation. Then, we look at the technique of frame selection, which can be seen as a combination of those two ideas. Finally, we look at prior

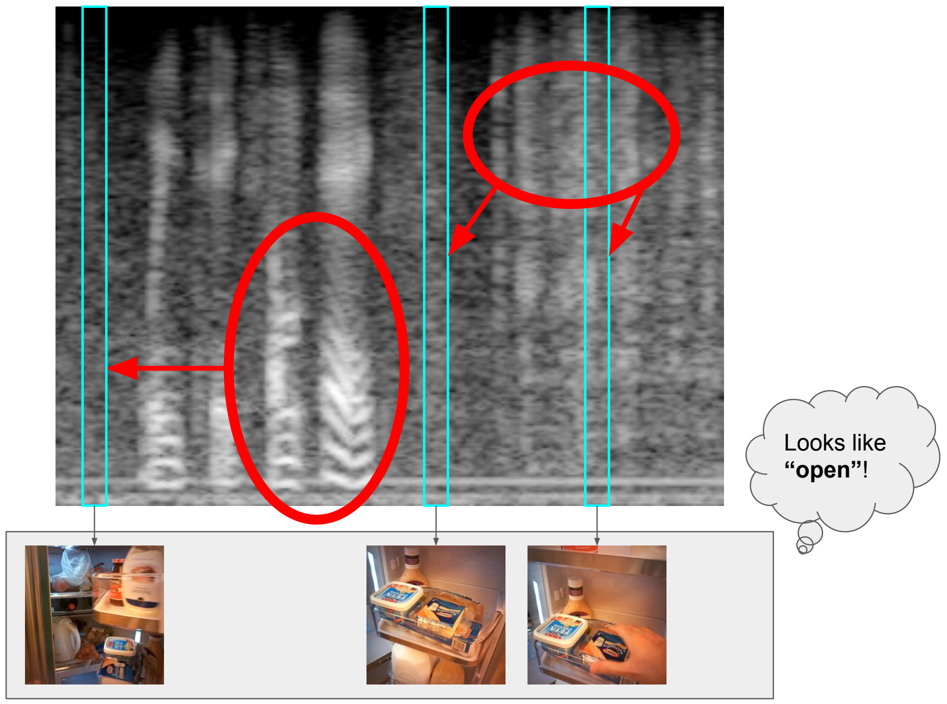


Figure 1: A high-level overview of our approach. Our method tries to learn to associate patterns in the audio data (marked in red) with locations of relevant frames (marked in blue). This results in a sparse set of frames to process later, but one which is more useful than a naive sampling.

work which has utilized audio for video processing.

2.1 Action Recognition

In the task of action recognition, a video clip is provided containing an action from some predetermined set of action classes, and the goal is to classify the action within the clip.

There have been many approaches to action recognition [11]. While earlier methods include reconstructing a 3D model or handcrafting relevant features, newer methods focus on learning useful features with deep neural networks. Karpathy et al. [14] was one of the first works to apply CNN techniques to a large-scale video dataset. Simonyan et al. [23] introduce a *two-stream* model, where some parts of the network process RGB frames and others process optical flow.

There have been many improvements to and variants of this basic technique. For example, some models replace some or all of the standard 2D convolutions with 3D convolutions [28, 3, 37]. Others extend the two-stream approach by looking at different modalities such as compressed motion vectors [39]. Models have also been proposed which attempt to use the long-range temporal structure of videos [32, 31].

Another related area of study is that of early event detection, where the goal is to detect an action as early as possible. Specifically, the model takes as input streaming video, and the goal is to make the latest frame that must be processed by the model as early as possible. Prior work on early event detection [12, 22, 5] requires processing a dense set of frames, which can be computationally demanding.

The above methods for action recognition are designed to maximize performance, without any constraints on computation time. To try to reduce computation, models have also been proposed to avoid recomputing intermediate features. For example, Zhu et al. [41] use a network to estimate the flow field, which is used to propagate features computed at key frames. Pan et al. [20] use sparsity to reduce the computation required to update intermediate features.

Like the works above, we use convolutional networks to process video, but our focus is on reducing computation. Unlike existing methods [41, 20], which still try to approximate running a CNN over the entire video, we instead choose only a subset of frames to process.

2.2 Adaptive Computation

Our work relates closely to the idea of adaptive computation, where the time allocated to a sample can depend on the current computational budget or the sample itself. This idea has background in cascaded classifiers as in [30]. Graves [10] adapts this idea for recurrent neural networks, letting the RNN learn when to stop the computation. Further works have looked at deep networks which can, in the middle of their computation, decide to follow alternate computation paths or terminate early. Some network designs conditionally execute blocks based on the current features [33, 29], or directly based on the input image [35]. Figurnov et al. [9] generalize this idea by not only conditionally executing entire blocks, but for each block, running different numbers of layers in different parts of the image. Some networks instead have a discrete set of computational paths, which are dynamically selected based on the current features [27, 13, 17]; this is termed "early exiting".

Our method shares similarities to models which conditionally execute blocks based on the input image, like BlockDrop [35]. However, instead of dropping blocks of a network, we drop frames of a video, which can be seen as orthogonal to blocks. Additionally, because video data is a 3D tensor, which is far too expensive to process at once, we use instead audio data to produce the block/frame selection policy.

2.3 Frame Selection

One common approach to video-related tasks such as recognition and localization is to choose only a subset of frames to process. While this can be done by choosing evenly-spaced frames, recent approaches have used recurrent neural networks to sequentially select frames based on the video clip itself [24, 38, 7, 36].

Su et al. [24] explore dynamically choosing which features of a network to compute based on the input video stream, which can include skipping frames entirely. They term this idea "feature triage". Notably, they treat the data both in the batch and streaming settings; in the latter, the model is constrained to a buffer of recent frames, and so cannot jump arbitrarily backwards in the video.

Similarly, Yeung et al. [38] introduce a recurrent model for action localization, which consists of a recurrent network that "jumps around" the video. Each iteration, the model looks at a frame and either decides to stop (outputting a prediction) or continue (outputting the position of the frame to look at next). The location of the next frame is unconstrained; at any iteration the agent can decide to move to any frame. Yeung et al. note that their model achieves state-of-the-art results while only using 2% or less of the frames. Fan et al. [7] use a similar recurrent model, but they constrain the next frame to be drawn from a discrete set of offsets (biased toward the forward direction).

Wu et al. [36] introduce a "global memory", which consists of the frame data, but spatially and temporally downsampled and processed with a small network. At each iteration, the agent queries the entire global memory using its internal state. The global memory acts as a summary of the entire video, allowing the agent to understand roughly what is going on in the rest of the video without having seen it.

These methods use only the video data to choose which frame to look at next. In contrast, our method uses audio data to determine which visual frames to analyze. While we select the subset of frames to process at once, we discuss possible generalizations to select more frames after obtaining additional information from previous frames.

2.4 Audio-visual Processing

Prior works have also incorporated audio data for video processing tasks such as action recognition [2, 18, 19, 34, 1, 26]. However, these methods all convert the audio data into feature vectors, and perform fusion to merge the audio and video streams into a single output.

Instead, we note that for a particular clip, the audio stream can be processed relatively cheaply, while processing all frames is expensive. We always do the former, but instead of unconditionally processing every frame (or a fixed subset), we conditionally process frames based on the results of processing audio.

3 Approach

We first formulate the problem of action recognition. Next, we describe the components of our model, and how they interact to classify video clips. Finally, we describe the steps to train our model.

3.1 Problem Formulation

Let a video clip s consist of video data V and audio data A. We assume that the video data V is a sequence of frames $\{f_1, f_2, ..., f_T\}$. In principle, no assumption needs to be made about the structure of the audio data A other than preserving the temporal location of audio events; for example, A could be a waveform, spectrogram, or mel-frequency cepstral coefficients (MFCCs), but it should not be, say, a ConvNet-processed feature vector of a spectrogram. In our experiments we use the spectrogram representation. Let $\mathcal{C}$ denote the set of possible classes of a video clip. The goal of action recognition is then: given a video clip $s = (V, A)$, output the correct class $c \in \mathcal{C}$.

Note that unlike most prior works on action recognition, we introduce an additional modality (audio) to our task. Other works which do use audio treat it as a parallel to the video data [2, 18, 19, 34, 1, 26]. Instead, we recognize that the entire audio data can be represented with a small amount of data, and treat it as an easy to process summary. (Concretely, a 10 second raw audio waveform sampled at 16000 Hz is the equivalent of a single 400×400 gray-scale image.)

3.2 Model

Our high level goal is to use audio data to intelligently select frames from the video data.

This raises the question: how effective can audio be in summarizing a video clip? We claim that many actions often will cause some sort of sound, which can be detected on a suitable representation. Moreover, different sounds can be distinguished, resulting in the potential to select frames differently depending on the

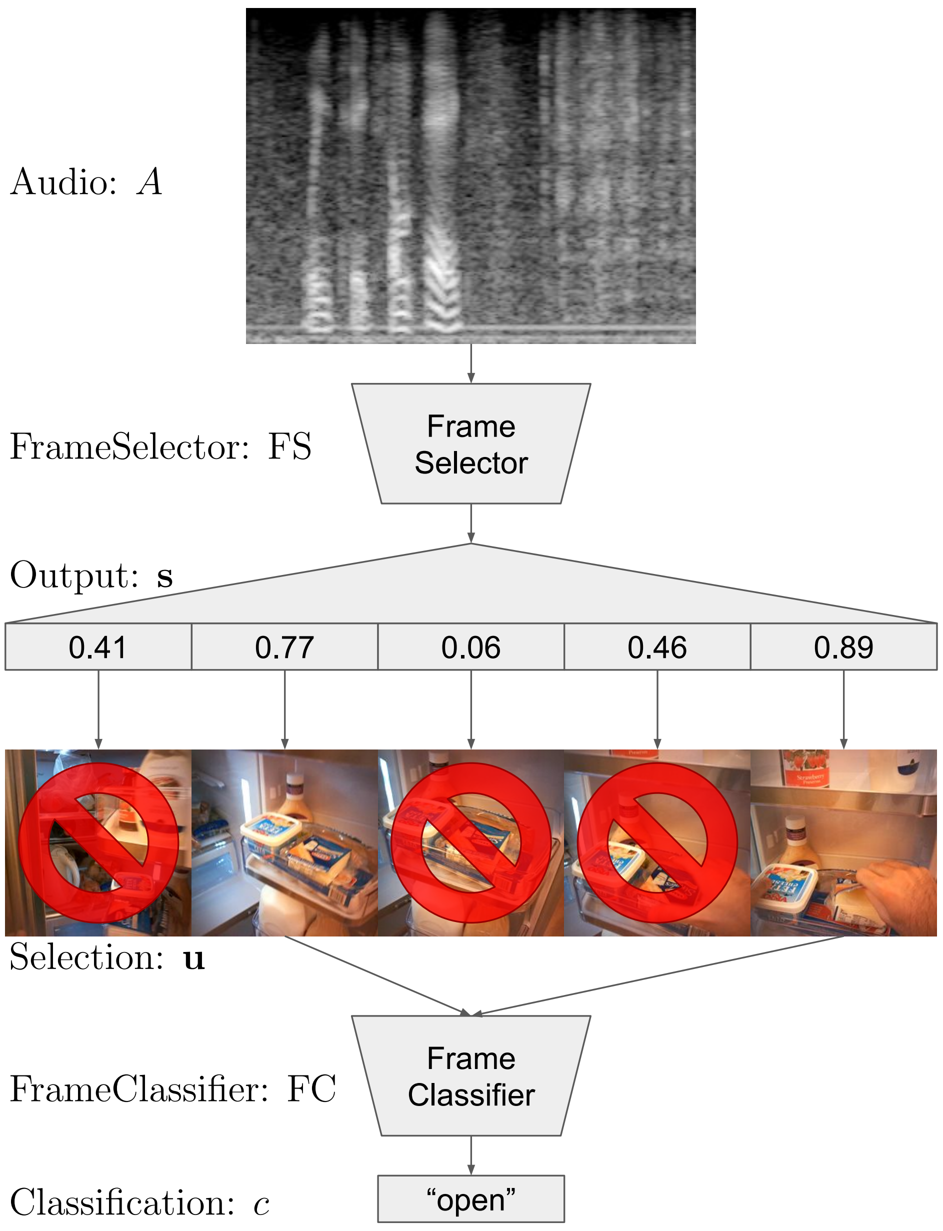
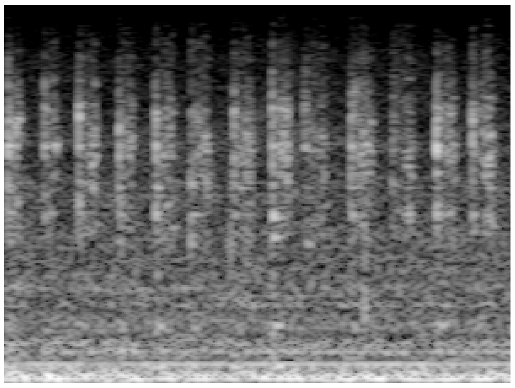


Figure 2: Summary of our proposed model. In order to process a video clip, a network (termed the *FrameSelector*) first processes the audio data as a spectrogram. Then, a separate network (termed the *FrameClassifier* processes the frames that were selected in order to produce the final classification.


(a) A spectrogram of a clip labelled “mix”

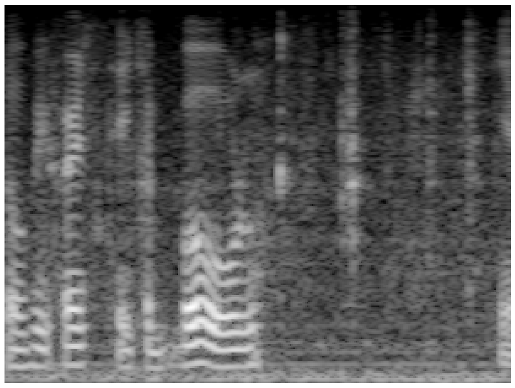

(b) A spectrogram of a clip labelled “take”

Figure 3: Some example spectrograms. As noted in the text, there are features in the spectrograms which correspond to actions in the video, such as the repeated vertical lines in Figure 3a and the “wavy” lines in Figure 3b.

sound itself; this includes selecting more frames when the model is less confident, or selecting frames from different offsets of the sounds that were detected. This is in contrast to selecting frames based only on the volume of a sound (though this can be quite effective in practice, for certain use-cases).

As an example, consider the act of closing a door. When a door slams shut, it usually causes a loud sound to occur. As closing a door is not the only action that can produce such a sound, it would be useful to check the frames corresponding to such a sound when it is noticed. Related to doors, another distinctive sound is a person knocking on a door. This often indicates that a door will be opening soon. However, since the door only opens after the knocking occurs, it would be inadvisable to look only at the frames when someone is knocking. Instead, it would be far more effective to look a few seconds after the knocking sound stops, to determine whether the door did actually open.

Concretely, we use the spectrogram representation of audio, which plots intensity as a function of frequency and time; in other words, it shows how much power is in a particular frequency for a given time segment. The conventions for displaying a spectrogram are as follows. The left-right axis corresponds to time, with time increasing toward the right. The up-down axis corresponds to frequency, with frequency increasing toward the top. The value of each entry corresponds to intensity or power, with the value increasing or the entry becoming brighter as intensity increases.

In Figure 3 we provide some spectrograms as examples. In Figure 3a, there are repeating vertical streaks throughout the entire video clip. This sound was produced by a utensil repeatedly hitting a container as it was used to mix. In general, if a similar pattern appeared in only part of the spectrogram, it may be wise to select frames from there (depending on the available action classes). In Figure 3b, on the left side, there are rows of “wavy” lines. These corresponded to talking; in general, sounds with a well-defined pitch correspond to patterns like this. This particular sound is not a single horizontal line due to overtones. Since “talking” is not a valid class in the datasets we used, the network would have to ignore patterns like this.

Our goal is to detect relevant patterns on the spectrogram, and select useful frames based on this information. Specifically, we expect that the audio will help determine where in the video frames should be processed, as in the door example above. Notably, the audio will not in general be enough to classify the action (although it can happen with a dataset that has very few classes, or has a very narrow scope).

Our model consists of two parts: a *FrameClassifier*, which classifies sequences of frames, and a *Frame-Selector*, which selects a subset of frames from the audio data. These are often abbreviated FC and FS respectively, and a particular selection of frames is termed a "policy". This terminology comes from reinforcement learning; in our primary method of training the FrameSelector, we treat the FrameSelector as a reinforcement learning agent, where the FrameSelector produces a selection of frames (policy) and gets a reward based on the result. As shown in Figure 2, the frames selected by the FrameSelector are processed by the FrameClassifier, which produces the final prediction.

3.2.1 FrameClassifier

Before even considering selecting frames, we first need a model to classify sequences of frames. We define a FrameClassifier as a network which takes as input a sequence of frames $\{f_1, f_2, ...f_k\}$ and outputs a classification $c \in \mathcal{C}$. In this case, we write $c = \mathrm{FC}(\{f_1, f_2, ...f_k\})$.

In some models like [38], in addition to the actual frame data, the network which processes frames also receives the original location of the frame in the video clip. However, we do not do this; once a frame is selected from a video clip, it is treated the same regardless where in the clip is it from. Only the ordering of the frames is preserved when fed into a FrameClassifier (and in fact, simple FC models may discard the order information as well). The intuition behind this choice is that it is possible to classify a video given just a few key-frames, without knowing how much space is in between. In fact, prior work has shown that key-frame selection is an effective approach for action recognition [40, 16, 15]. Of course, one gains much more information about the video given all the frames, but they are certainly not all necessary.

In our particular implementation, the FrameClassifier consists of the trunk of a ConvNet (specifically, a ResNet), followed by a series of fully-connected layers. In the middle of the fully-connected layers, we average-pool features across the dimension corresponding to the input frames in order to account for varying numbers of frames.

3.2.2 FrameSelector

Once we have a FrameClassifier, we can classify video clips. However, we do not know anything about the video clip before providing it to the FrameClassifier, so we are limited to static policies. For example, taking all frames is very inefficient in terms of computation time, while taking only a few frames risks missing important frames which occurred in between. Even a compromise, such as taking every x frames, may end up choosing frames which are not part of the action, or where no useful information is being displayed in the video data.

Therefore, we use the audio data to choose this policy. As noted above, there are often characteristic patterns in the spectrogram representation which can indicate particular actions, which we would like our frame selection network to recognize.

More formally, we define a FrameSelector as a network which takes as input audio data A and outputs a length T vector $\mathbf{u}$ where $\mathbf{u}_k$ is 1 if the kth frame is selected, and 0 if it is not.

In our particular implementation, the FrameSelector consists of the trunk of a ConvNet (specifically, a ResNet), followed by a series of fully-connected layers.

3.3 Training

In order to ensure our model trains, we split the training process into two stages: FrameClassifier-only and FrameSelector-only. In these stages only one network is trained at once. We also experiment with two additions to the training process. The first is joint finetuning, where both networks are trained simultaneously after the FrameSelector-only training; this is to finetune the FrameClassifier for the particular distribution of frames the FrameSelector chooses, as opposed to the arbitrary one during the FrameClassifier-only stage. The second is volume pretraining, where the FrameSelector is pretrained to predict volume; this is to encourage the FrameSelector to produce different outputs based on the input, and not just output a constant policy.

3.3.1 FrameClassifier-only

Initially, we have neither a working FrameClassifier nor a working FrameSelector. Just to be able to classify video clips, we first need to train a FrameClassifier. However, we importantly cannot use a FrameSelector's policies (much less train it), since we have no guarantee that it selects any useful frames. Accordingly, this stage of training is called "FrameClassifier-only".

Because we do not have a working FrameSelector yet, we must choose some distribution over frames for each sample to provide to the FrameClassifier. Given video data $V = \{f_1, f_2, ...f_T\}$, we independently choose each frame f_i with probability p if it has the same class as the sample (we assume we are provided with either action extents, or per-frame labels), and probability 0 otherwise. For example, if $p = 1/2$, each matching frame is selected with probability $1/2$. If $p = 1$, all matching frames are selected for each sample.

Originally, the value of p was chosen uniformly from $[0, 1]$ independently for each sample. This was intended to increase the robustness of the FrameClassifier for later, where the number of frames selected could potentially vary significantly. However, we found that fixing the p at 1 for all samples improved later performance, despite not matching exactly the environment of the FrameSelector.

When training the FrameClassifier, we use the standard cross-entropy loss on the output. Specifically, given a sample $s = (V, A)$ and class c, if the indices which were selected are $\{i_1, i_2, ...i_k\}$, we compute the FrameClassifier output to be $c' = \text{FC}(\{f_{i_1}, f_{i_2}, ...f_{i_k}\})$ and the cross-entropy is computed with the prediction c' and the label c.

3.3.2 FrameSelector-only

Now that we have a working FrameClassifier, we can begin to train the FrameSelector. As noted above, since the FrameSelector is initially untrained, it may not be safe to update the FrameClassifier at the same time, so we train only the FrameSelector. Accordingly, this stage of training is called "FrameSelector-only".

Unfortunately, the selection process of the FrameSelector is not differentiable. This is due to the act of adding or removing a frame being a discrete change. Therefore, we use policy gradient [25, 6] to train the FrameSelector.

For a given policy $\mathbf{u}$, the reward is defined as

$$R(\mathbf{u}) = \begin{cases} 2^{-|\mathbf{u}|_0/5} & \text{if correct} \\ -\gamma & \text{otherwise} \end{cases} \tag{1}$$

where γ denotes the penalty. If the model (FrameSelector and FrameClassifier) produces an incorrect classification, we give it a reward of $-\gamma$, penalizing the incorrect classification. However, if it is correct, we give it a positive reward based on how many frames it selected. This reward is scaled so it halves every 5 additional frames.

It may seem that the positive reward decreases far too quickly to be useful. For example, one might consider instead a linear scaling, with one frame selected giving reward 1, and all frames selected giving reward 0. However, the difference between the rewards for 1 frame selected and (say) 5 frames selected would be the same as that between all but 1 frames selected and all frames selected. We only plan to select a few frames, so using this exponential function allows us to increase the change in reward at a low number of frames.

Also note that this function considers the number of frames selected, not the proportion of frames selected. This is due to several reasons. One reason is that video clips may vary in FPS, so a 3 second clip may have 72 frames in one dataset or 180 frames in another; however, the extra frames are highly correlated and hence provide little to no extra information about the scene itself. Another reason is we assume that one clip contains exactly one action instance (as opposed to an action localization task), so it is never necessary to spread selections across multiple parts of the clip to account for multiple action instances.

We base the reinforcement learning process on that of BlockDrop [35]. Our policies are drawn from a T-dimensional Bernoulli distribution, so given an input audio data A, the distribution of policies is

$$\pi_{\mathbf{W}}(\mathbf{u}|A) = \prod_{k=1}^{T} \mathbf{s}_k^{\mathbf{u}_k}(1-\mathbf{s}_k)^{1-\mathbf{u}_k} \tag{2}$$

$$\mathbf{s} = \mathrm{FS}(A; \mathbf{W}) \tag{3}$$

where $\mathbf{W}$ represents the weights of the FrameSelector, and $\mathbf{s}$ represents the output of the FrameSelector after the sigmoid. We define the expected reward J as

$$J = \mathbb{E}_{\mathbf{u}\sim\pi_{\mathbf{W}}}[R(\mathbf{u})] \tag{4}$$

which represents the value we wish to optimize. The gradients can be defined as

$$\nabla_{\mathbf{W}} J = \mathbb{E}[R(\mathbf{u})\nabla_{\mathbf{W}} \sum_{k=1}^{T} \log[\mathbf{s}_k\mathbf{u}_k + (1-\mathbf{s}_k)(1-\mathbf{u}_k)]]. \tag{5}$$

Furthermore, we utilize a self-critical baseline $R(\tilde{\mathbf{u}})$ as in [21], which lets us rewrite the above equation as

$$\nabla_{\mathbf{W}} J = \mathbb{E}[A\nabla_{\mathbf{W}} \sum_{k=1}^{T} \log[\mathbf{s}_k\mathbf{u}_k + (1-\mathbf{s}_k)(1-\mathbf{u}_k)]] \tag{6}$$

where A, the "advantage", is defined as $A = R(\mathbf{u}) - R(\tilde{\mathbf{u}})$, and $\tilde{\mathbf{u}}$ is the maximally probable configuration under $\mathbf{s}$ (the "thresholded" policy where $\tilde{\mathbf{u}}_i = 1$ if $\mathbf{s}_i > 0.5$, and $\tilde{\mathbf{u}}_i = 0$ otherwise). The purpose of the self-critical baseline here is to reduce the variance of the gradient estimates. We use this technique as opposed to learning from a value network because training such a network is difficult in this context. Finally, to encourage exploration and avoid components of the policy from saturating to 0 or 1 (never or always selecting a frame), we also introduce the α parameter to bound the distribution induced by $\mathbf{s}$. This results in the distribution

$$\mathbf{s}' = \alpha \cdot \mathbf{s} + (1 - \alpha) \cdot (1 - \mathbf{s}), \tag{7}$$

which is bounded in the range $1 - \alpha \leq \mathbf{s} \leq \alpha$. The actual policy is sampled from the modified distribution $\mathbf{s}'$.

In summary, in order to train our FrameSelector, we treat the output vector $\mathbf{s}$ as a T-dimensional Bernoulli distribution. We apply the α parameter, resulting in $\mathbf{s}'$. We then sample from $\mathbf{s}'$, resulting in a sampled policy $\mathbf{u}$, and we threshold $\mathbf{s}$ (or equivalently $\mathbf{s}'$), resulting in a thresholded policy $\tilde{\mathbf{u}}$. The advantage A is treated as the gradient, which is used to update the weights of the FrameSelector.

3.3.3 Joint Finetuning

Now we have both a trained FrameClassifier and a trained FrameSelector. However, there is a mismatch between the sets of frames the FrameClassifier saw during the FrameClassifier-only stage, compared to the sets of frames provided by the current FrameSelector. For example, the FrameClassifier has been trained on all frames, including those which may be useless (for example, those where the camera was looking away from useful information). However, the FrameSelector may have learned to avoid selecting these frames. In order to reduce the impact caused by this effect, we now jointly train both the FrameClassifier and the FrameSelector. Accordingly, this stage of training is called "joint finetuning" (after the corresponding stage in BlockDrop [35]).

Because the FrameClassifier and FrameSelector contribute independent loss terms, we just need to add both loss terms when computing gradients. In effect, we run the FrameClassifier-only and FrameSelector-only stages simultaneously.

3.3.4 Volume Pretraining

We also look at adding a pretraining stage for the FrameSelector. Instead of starting the FrameSelector-only stage with a randomly initialized FrameSelector, we pretrain it to select frames with high "visual volume"; that is, frames corresponding to high-intensity parts of the spectrogram. We use this method, as opposed to the actual physical volume, to match the input provided to the FrameSelector model. Importantly, a network can compute visual volume by summing over the columns of the input spectrogram, so this behavior is in theory learnable by a FrameSelector. The main difference with actual volume is that the entries of our spectrograms have been log-scaled and normalized, which weights the high-intensity pixels less. However, this is countered by the fact that most sounds, even pure tones, tend to be spread out over multiple frequency bins on a spectrogram.

In some sense, we train the FrameSelector to approximate a baseline which selects frames based solely on the visual volume of the spectrogram.

The goal of volume pretraining is to encourage the FrameSelector to use the provided audio data, instead of ignoring the input and outputting a constant policy. It does so by encouraging the FrameSelector to learn to associate parts of the input spectrogram (such as high-intensity columns) with corresponding parts of the output vector (high expected outputs). Since our FrameSelector topology discards temporal information after computing the final filter responses, this helps the FrameSelector learn correspondences between the features and their temporal location for the final layer.

We look at two volume pretraining methods:

- Regression: We treat pretraining as a regression problem, where the actual volume is compared to the output of the FrameSelector.
- Classification: We treat pretraining as a classification problem, where a frame is considered “loud” if its volume is above the mean; the FrameSelector then independently classifies each frame as loud or not.

3.3.5 Summary

To summarize, to train an instance of our model, we start with a randomly initialized FrameClassifier and FrameSelector. As the first step, we run the FrameClassfier-only stage to train the FrameClassifier. Optionally, we run the volume pretraining stage to pretrain the FrameSelector. (Of course, these two stages can be done in any order since they only work with one network.) We then run the FrameSelector-only stage to train the FrameSelector to work with the FrameClassifier. Finally, we optionally run the joint finetuning stage to finetune the FrameClassifier to the precise distribution of frames generated by the FrameSelector.

4 Results

We first describe the datasets we used, the baselines we compare our method against, and the specific implementation details for our experiments. Then, we compare the basic version of our approach with the baselines, show examples produced by our method, and propose reasons for the results. Finally, we look at the performance of variants of our approach.

4.1 Datasets

To evaluate our method, we consider two video datasets which satisfy several criteria. First, the dataset must contain audio that is the actual sound from the activities depicted; in particular, videos with their audio replaced with music, as in some datasets, would not work. Also, each action should preferably be at least a certain “reasonable” length, and actions should be well-separated from each other if possible (with a similar “reasonable” length scale). We encourage the former because actions which are too short may only have a few frames which correspond to the actual action class, making it very hard to exactly select frames where the action is occurring. We encourage the latter because if two actions are very close together, we cannot create extents which contain only the start or end of one action, which could potentially appear at test time. However, in principle, given enough data and computation time the algorithm could train when neither of these conditions hold; they are not fundamentally required for the method to work.

While these criteria do not imply it has to be the case, both datasets ended up being egocentric cooking datasets. While the setting and many of the action classes are common (such as “open” and “close”), the datasets will have varying quality, as well as different tasks performed in different locations. Also, egocentric video has the benefit that sounds recorded are those that the actor hears, where a third-person video may record irrelevant sounds which happen to be close to the audio recording device.

For both datasets, we took the 20 most common actions by frequency. Furthermore, for both datasets, some action annotations partially or entirely overlapped with other annotations in time. Any such action was removed. Properly handling such overlapping data would require extending our method to a detection paradigm, which we leave as future work.

In order to identify valid training samples, we look at every possible extent of length 3 seconds (aligned to frames). We keep an extent if the most common action across all the frames (excluding no-action frames) had an 80% majority, and had least 0.5 seconds worth of frames of the majority action.

Note that each action instance can have multiple extents corresponding to it. To deal with this, we use a form of data augmentation. At train time, each action instance (sample) is randomly offset while preserving the conditions above. At test time, the offset is chosen to be the middle of the possible offsets. Note that this does not necessarily center the action instance within the extent.

4.1.1 GTEA Gaze+

For our first dataset, we use the Georgia Tech Egocentric Activity Gaze+ (GTEA Gaze+) dataset [8].[1] The GTEA Gaze+ dataset consists of 27 egocentric videos of six people cooking one of seven meals. The videos are on average 10.30 minutes long. After pruning, there are 1765 train samples and 239 test samples. The distribution of actions is shown in Table 1a.

4.1.2 EPIC-KITCHENS

For our second dataset, we use the EPIC-KITCHENS dataset [4]. The EPIC-KITCHENS dataset consists of 272 egocentric videos of 28 people performing cooking-related tasks. The videos are on average 8.75 minutes long. After pruning, there are 9989 train samples and 2994 test samples. The distribution of actions is shown in Table 1b.

4.2 Evaluation

To evaluate a frame selection method, we plot the accuracy or mean average precision (mAP) against the average number of frames selected per sample for the test set. Each instance of our model (FrameClassifier and FrameSelector) corresponds to a point on this graph. However, we would also like to see the effect of selecting more frames on the performance of the model. Thus, we train multiple versions of our model, varying the penalty γ in Equation 1. If any instance of our model selects less than 1 frame on average, we assume it has reached a degenerate state, and do not plot it. This occurs when the penalty is set very low, causing the FrameSelector to prefer selecting no frames (which is always counted as an incorrect output) instead of selecting even a single frame.

We also present some baselines which require a preselected number of frames (defined below). For example, Random chooses k random frames, for a predetermined value of k. As with our model, we can "train" multiple versions of such a baseline by simply varying the number of frames taken. In fact, because we directly control the exact number of frames selected by the baseline, we can easily generate data points

[1] We downloaded the dataset from `http://ai.stanford.edu/~alireza/GTEA_Gaze_Website/GTEA_Gaze+.html`, which contains slightly different data from `http://www.cbi.gatech.edu/fpv/`.

Rank	**Action name**	**# clips**
1	take	517
2	put	405
3	open	274
4	close	201
5	cut	107
6	move around	91
7	transfer	73
8	pour	65
9	turn on	47
10	read	46
11	mix	35
12	turn off	28
13	squeeze	24
14	distribute	21
15	compress	18
16	flip	15
17	spread	11
18	dry	9
19	peel	9
20	wash	8

(a) for GTEA Gaze+

Rank	**Action name**	**# clips**
1	take	3043
2	put	2963
3	open	1466
4	wash	1414
5	close	874
6	cut	484
7	mix	477
8	pour	401
9	move	236
10	turn-on	235
11	throw	220
12	remove	217
13	turn-off	189
14	dry	186
15	insert	132
16	turn	127
17	peel	106
18	shake	83
19	press	67
20	squeeze	63

(b) for EPIC-KITCHENS

Table 1: Action distributions: for each action class, how many instances of that class are in the dataset. Note that due to the random-offset augmentation, there are multiple valid clips within an action extent, and so the number of possible clips seen at train time is more than the number here.

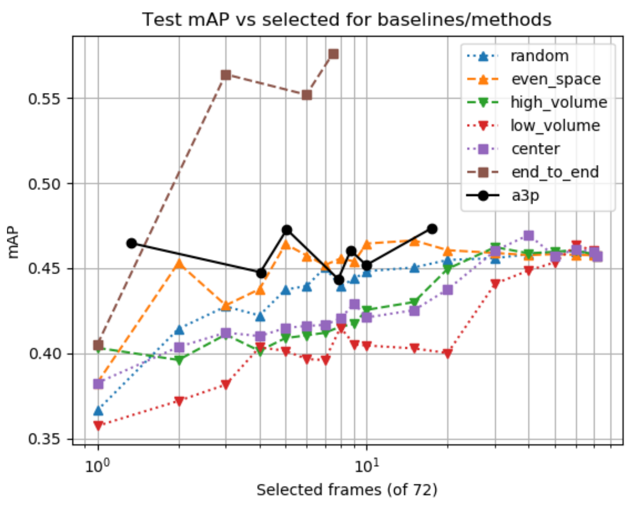
for any possible x-coordinate. In contrast, we can only indirectly control the number of frames selected for our actual model, by changing γ.

4.3 Baselines

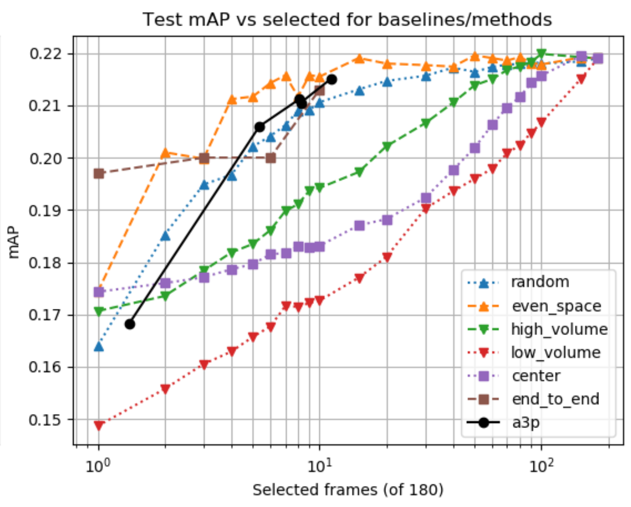
To evaluate our method (which we denote A3P, from "Audio as a Preview"), we consider a few simple baselines for frame selection (listed below), as well as a state-of-the-art method for frame selection that uses visual frames with a recurrent neural network [38] (which we denote EndToEnd, from the first few words of the title). Note that all baselines here except EndToEnd require preselecting the number of frames, which we denote k.

- Random. k random frames are selected. This baseline is the simplest possible. Importantly, it makes no assumptions about the position of actions within a video clip. For example, EvenSpace (discussed below) will consistently fail at both $k = 1$ and $k = 2$ if the only useful frame is 3/8 through the clip, whereas Random still has a chance of choosing this frame.
- EvenSpace. k evenly-spaced frames are selected. Specifically, we partition the video clip into k blocks, and select the middle frame in each block. For example, if $k = 1$, the middle frame is selected; if $k = 2$, the 1/4 and 3/4 frames are selected; if $k = 3$, the 1/6, middle, and 5/6 frames are selected; etc. Unlike Random, EvenSpace deterministically selects frames, so it is guaranteed not to select frames which are close to each other. This gives the FrameClassifier frames across the entire extent, which would give it an increased chance of finding a useful frame.
- HighVolume. The k frames whose columns in the spectrogram have the highest intensity are selected. As noted above, this "visual volume" does not exactly correspond to physical volume, but it has the advantage that it can be derived easily from the same spectrograms passed to the FrameSelector. Additionally, this method avoids having to deal with unequal perception of volume across different frequencies. The purpose of this baseline was to test whether actions can be found solely using loud noises. Specifically, sudden unpitched loud noises correspond to bright vertical columns in a spectrogram, as the sound is localized in time, but not in frequency. As an example, if the only two action classes were "thunder" and "car passing", then selecting only loud frames would select frames which mostly likely belong to one of these classes, allowing for a successful classification.
- LowVolume. The k frames whose columns in the spectrogram have the lowest intensity are selected. This baseline is a parallel to HighVolume; it was expected that low volume frames would usually be useless in classification, as the camera would most likely be looking away from any useful objects.
- Center. k frames in the center are selected. Originally, all action instances were centered within the extents, so frames in the middle of the extent were guaranteed to be the correct class. This caused the naive Center baseline to perform quite well. However, after adding the random-offset augmentation, the center of the extent is no longer guaranteed to be the correct class, although correct frames are still biased toward the center. Now Center can be seen as evaluating whether this center bias is still significant.
- EndToEnd. Frames are chosen and classified by the model in [38].[2] Note that because this model already has a FrameClassifier-like component, we use the classifications it generates.

[2]Code adapted from `https://github.com/ShangtongZhang/DeepRL`.



(a) for GTEA Gaze+



(b) for EPIC-KITCHENS

Figure 4: mAP versus frames selected for datasets - baselines. Random is averaged over 9 trials. Our model performs about as well as high-scoring baselines such as EvenSpace. The untrained baselines fall into two groups, based on the policies they produce. Namely, the baselines which select "scattered" frames (Random, EvenSpace) tend to have a better performance than baselines which select "clumped" frames.

4.4 Implementation Details

In our experiments, we let the frames be represented as 224×224 RGB images, and we let the audio be represented as a spectrogram.

For our spectrograms, we use 224 logarithmically-spaced frequency bins. We let each column of the spectrogram represent 0.01 seconds; since each video clip was 3 seconds, this means each spectrogram is a 300×224 grayscale image.

For the FrameClassifier, we use a ResNet-101 pretrained on ImageNet as a feature extractor. To process the features, we apply a fully-connected layer with activation, followed by an average pool over all frames, followed by a final fully-connected layer that produces the output class.

For the FrameSelector, we use a ResNet-18 pretrained on ImageNet as a feature extractor. To process the features, we apply a fully-connected layer with activation, and then another fully-connected layer that produces the output policy.

4.5 Action Recognition Performance

We first look at the overall performance of our model, as compared to the baselines. In Figure 4, we plot the mean average precision (mAP) against the number of selected frames.

A general contrast between the two datasets is that GTEA Gaze+'s graphs are noisier than EPIC-KITCHENS's. This is due to the fact that GTEA simply has fewer samples. Accordingly, to verify that a "trend" in the graph is not just due to random noise, we often check with the EPIC-KITCHENS graphs.

First we note some interesting peculiarities with the untrained baselines, to help understand A3P's position among them.

For most of the trials, the untrained baselines compare as follows:

$$\textsc{EvenSpace} > \textsc{Random} > \textsc{HighVolume} > \textsc{Center} > \textsc{LowVolume}$$

This is especially clear in the EPIC-KITCHENS graph.

We note that the untrained baselines tend to fall in two groups. One group consists of Random and EvenSpace; these are the policies which select "scattered" frames, and score extremely high on the graphs. The other group consists of HighVolume, Center, and LowVolume; these are the policies which select "clumped" frames, and score lower on the graphs. Interestingly, while HighVolume and LowVolume select frames in a similar manner, they have significant differences on the graph, indicating that volume can at least partially give an indication of actions. This supports our hypothesis that volume can correlate with how useful a frame is.

Now we compare our A3P. In both datasets, A3P outperforms four of the five untrained baselines (Random, HighVolume, Center, LowVolume), and it has roughly the same performance as EvenSpace. The performance of EndToEnd [38] relative to the other baselines varies significantly between the two datasets, so it is hard to directly compare our method to it in a way that generalizes across datasets. For GTEA Gaze+, EndToEnd significantly outperforms the rest of the baselines and our method. However, for EPIC-KITCHENS, EndToEnd has performance roughly between EndToEnd and Random; our method is even able to surpass EndToEnd slightly for certain amounts of selected frames.

4.5.1 Performance During Training

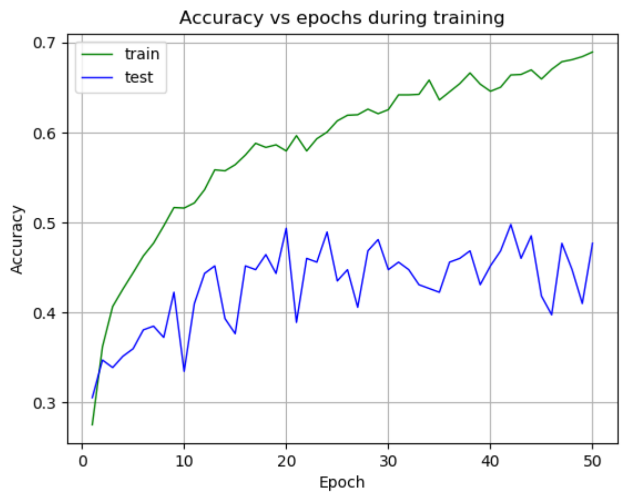
From the results above, it may seem that the number of samples is too small. To analyze this possibility, we look more closely at the train-time statistics. For these results we look at the network trained with penalty $\gamma = 5$, as the behavior of these networks is representative of those trained with different penalties.

As shown in Figure 5, during the FrameClassifier-only stage the test accuracy is able to increase. This indicates that we are not just learning random noise in our data, or overfitting immediately.

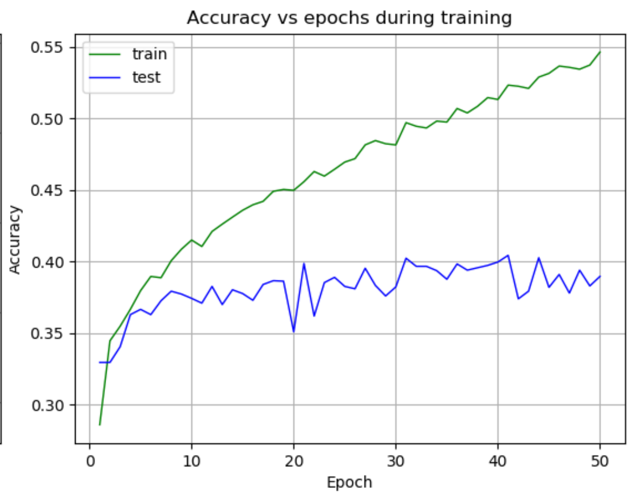
An interesting behavior occurs when the FrameSelector-only stage begins. As shown in Figure 6, the FrameSelector immediately starts to reduce the number of frames selected, which is expected. However, Figure 7 shows the accuracy of the system decreases while this occurs. One possibility is that the penalty is set too low, so the FrameSelector would rather get samples wrong than select more frames. However, when increasing the penalty, the number of frames selected stops decreasing at a higher value, but the accuracy still decreases or stays roughly the same.

Interestingly, the number of selected frames in EPIC-KITCHENS drops lower than for GTEA Gaze+, even though there are more frames in a video clip (due to FPS differences). This cannot be explained by EPIC-KITCHENS being easier to classify; as shown in Figure 4b, this is certainly not the case. However, it may be that for EPIC-KITCHENS, adding an extra frame at low numbers of selected frames does not increase the accuracy enough compared to the cost of the extra frames in the reward function. Alternatively, this might simply be a consequence of the larger number of samples in EPIC-KITCHENS, which results in more weight updates to the networks during training, effectively training the network more in each epoch.

Finally, note that the test accuracy often varies by up to 5% from one epoch to the next; this indicates that per-epoch variation in performance could have been a factor in the final measurements.

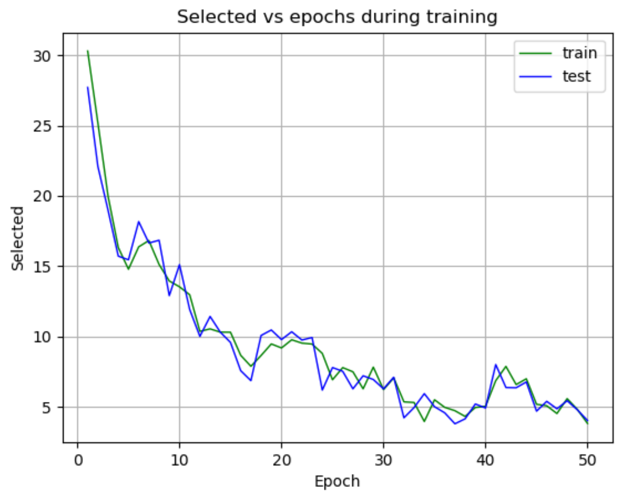


(a) for GTEA Gaze+

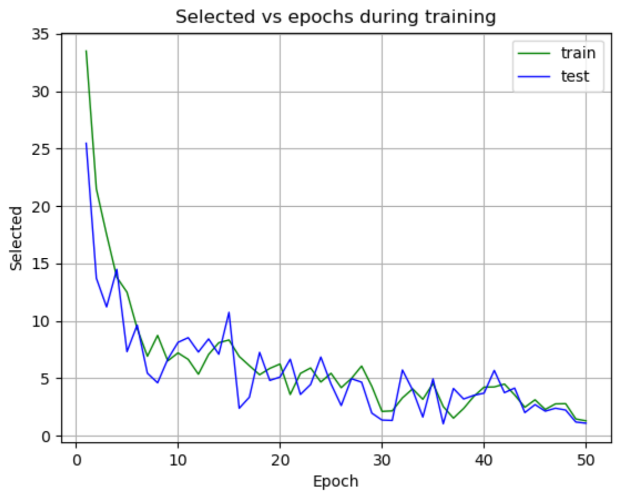


(b) for EPIC-KITCHENS

Figure 5: Accuracy versus epochs during FrameClassifier-only. While the test accuracy stops increasing at a certain point, it does increase for the first few epochs, indicating the FrameClassifier learns to generalize somewhat.

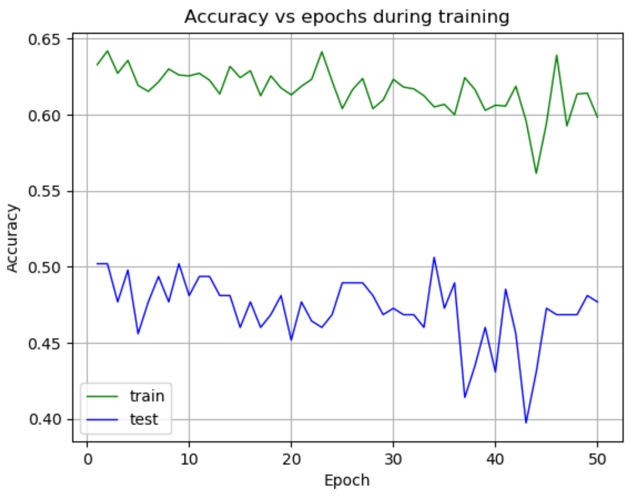


(a) for GTEA Gaze+

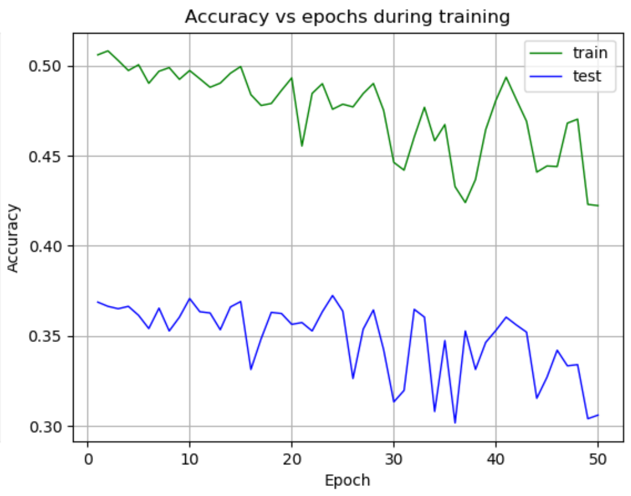


(b) for EPIC-KITCHENS

Figure 6: Selected frames versus epochs during FrameSelector-only. As the FrameSelector-only training progresses, the FrameSelector is indeed able to learn to select fewer frames. The results of training networks with varying penalties show that increasing the penalty causes the number of selected frames to stop decreasing at a higher value.



(a) for GTEA Gaze+



(b) for EPIC-KITCHENS

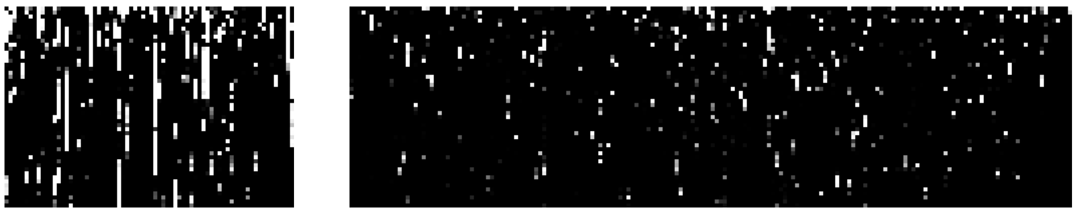
Figure 7: Accuracy versus epochs during FrameSelector-only. As the FrameSelector-only training progresses, the accuracy decreases slightly. The results of training networks with varying penalties show that increasing the penalty causes the accuracy to decrease by a smaller amount, or not at all.

4.5.2 Policy Statistics

We now verify to what degree the policies change based on the input. The goal was to have the FrameSelector dynamically choose frames based on the spectogram, so it could for instance select many frames on the left (earlier), or many frames on the right (later). First we present a "selection diagram" in Figure 8. The horizontal position represents frame position within the video clip (left is earlier), and the vertical position represents epoch (higher is earlier epoch, top is after epoch 1). The brightness of a square represents how many policies included a particular frame position within the video clip in a particular epoch. The policies used here are the thresholded outputs of the FrameSelector, which corresponds to how the output of a FrameSelector would be used at test time. We again present only the $\gamma = 5$ network since it is representative of the other networks.

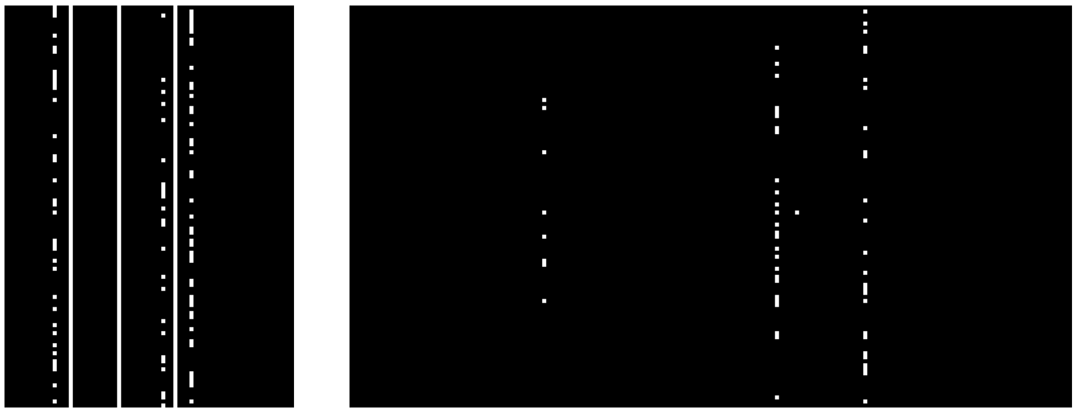
We can see clearly defined pure-white streaks in Figure 8a for GTEA Gaze+. This indicates that often during the training process, the FrameSelector would consistently select a particular frame position, but after another few epochs, it would "lose interest" and move on to other frame positions. There is a notable difference in Figure 8b for EPIC-KITCHENS. There are far more isolated pixels, although the streaks are still present. This may be because EPIC-KITCHENS has more samples than GTEA Gaze+, which causes the patterns in Figure 8a to be vertically compressed into single pixels.

Now we look closely at the particular collection of policies produced by our model after training. Instead of plotting the epoch along the vertical axis, we fix the epoch (to be the last one), and instead have the vertical position correspond to a sample. Thus each row is a 0-1 vector representing each sample's policy as chosen by the FrameSelector. One can view each row of Figure 8 as looking "down through" one of the following diagrams. Due to the size, we only show the policy visualization for 100 random samples. This is shown in Figure 9.



(a) for GTEA Gaze+ (b) for EPIC-KITCHENS

Figure 8: “Selection diagram”: what the average policy looks like at each epoch. The vertical position represents the epoch (training progresses from top to bottom), the horizontal position represents the position within the clip (earlier at the left), and the brightness represents the proportion of policies which chose that frame in the epoch. For several epochs, the FrameSelector would always select a particular frame, which is represented by the bright vertical streaks on the diagram. This effect is less pronounced for EPIC-KITCHENS.



(a) for GTEA Gaze+ (b) for EPIC-KITCHENS

Figure 9: “All policy visualization”: each policy (for 100 random samples). The policies generated by the FrameSelector seem to consist of only a few frames, with minor variation for certain frames.

Note that some columns (frame positions) are always selected, no matter the input spectrogram. Some columns are only sometimes selected; in fact, looking closely at the two rightmost “sometimes selected” columns in Figure 9a, one can see that those two frame positions are never selected at the same time. While it seems this indicates some amount of intelligent selection, such a behavior can be explained by a single node which has a positive weight on one position and a negative weight on the other.

However, the fact that the network converged onto an almost constant policy indicates that for the current problem definition, the optimal policy is indeed to select a constant set of frames. In this particular instance, GTEA Gaze+ had only 8 unique policies and EPIC-KITCHENS had only 19 unique policies, which is the equivalent of only selecting frames from some fixed set of 3 frames (5 frames for EPIC-KITCHENS).

4.5.3 Resulting Policies

In Figure 10, we give examples of the policies our model outputs, alongside the frames the selections correspond to. These particular examples are cases where our model produces the correct output. We only present these cases, because other than the final FrameClassifier output being incorrect, the behavior of the policies is very similar for samples our model misclassifies.

Looking at the GTEA Gaze+ sample in Figure 10a, the FrameSelector selected four frames. Visually these frames are very similar; all of them contain a hand in the process of picking up (“taking”) a bowl. This seems to indicate that the FrameSelector can still be improved, perhaps by selecting only one frame here. It is possible that the FrameSelector selects more frames to avoid the chance of selecting no useful frames.

In the EPIC-KITCHENS sample in Figure 10b, the FrameSelector selected two frames. Unlike above, these two frames are visually very different. The first frame has little to no indication of a potential “open” event, while the second does, indicating that the FrameSelector does sometimes select frames “just in case” one of them does not contain useful information.

In both cases, there is not an obvious correlation between the frames selected and the spectrogram data. This agrees with Figure 9, where the policies for all of the samples tended to use only a small fixed set of frames, instead of varying per sample.

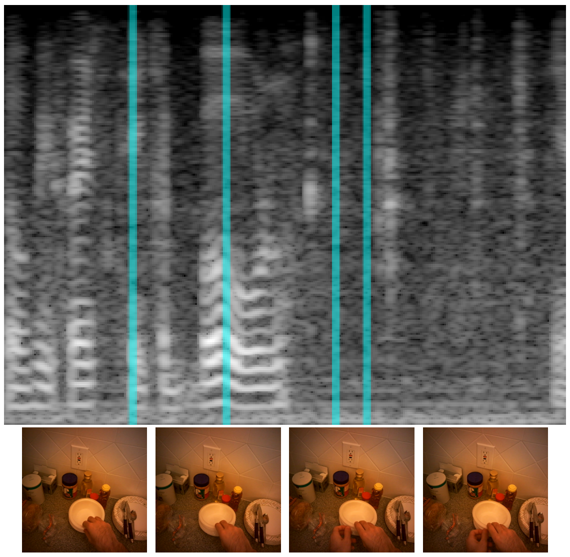
4.6 Variants

When testing our model, we experiment with several variants.

4.6.1 Joint Finetuning

In this variant, we added the Joint Finetuning stage after the FrameSelector-only stage. We show the effects of adding this stage in Figure 11.

The experiments indicate that joint finetuning can have varying effects. For GTEA Gaze+, at high numbers of frames, the performance of the model increases somewhat. In contrast, the performance of the model decreases slightly for EPIC-KITCHENS. The behavior in EPIC-KITCHENS is much easier to explain: the joint finetuning stage continues the trend of decreasing accuracy from the FrameSelector-only stage as seen in Figure 7b. The behavior in GTEA Gaze+ is not as clear; it may indicate that joint finetuning tends to favor policies with more frames selected, but since GTEA Gaze+ has so few samples, there is a distinct possibility that the increase in performance is just noise.


(a) for GTEA Gaze+; action class "take".

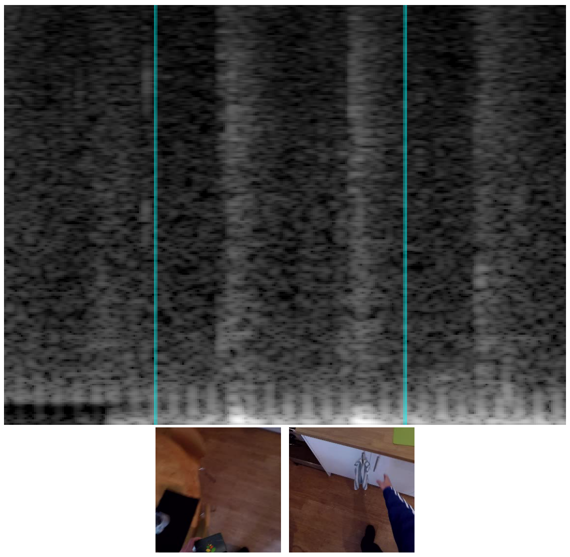
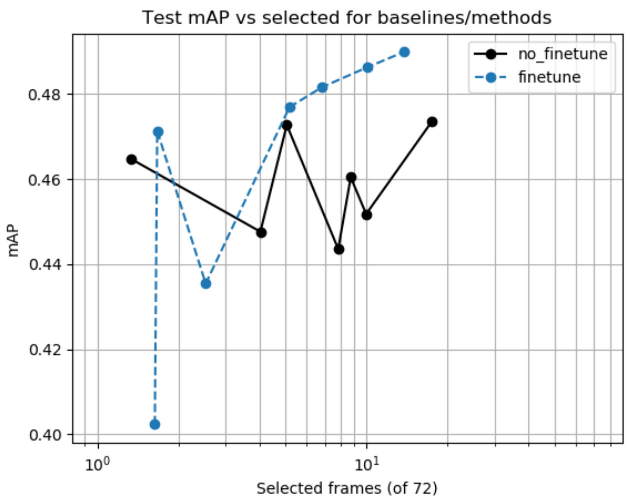
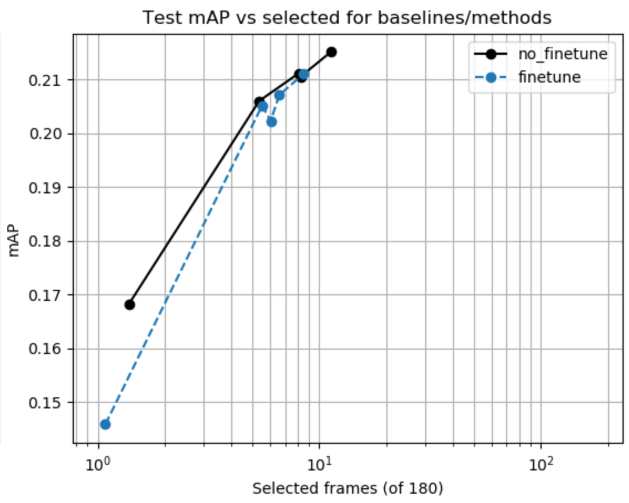

(b) for EPIC-KITCHENS; action class "open".

Figure 10: Examples of policies chosen by our method. The blue columns on the spectrograms represent the frames chosen by the FrameSelector; the widths are to scale (recall that EPIC-KITCHENS has a higher FPS).

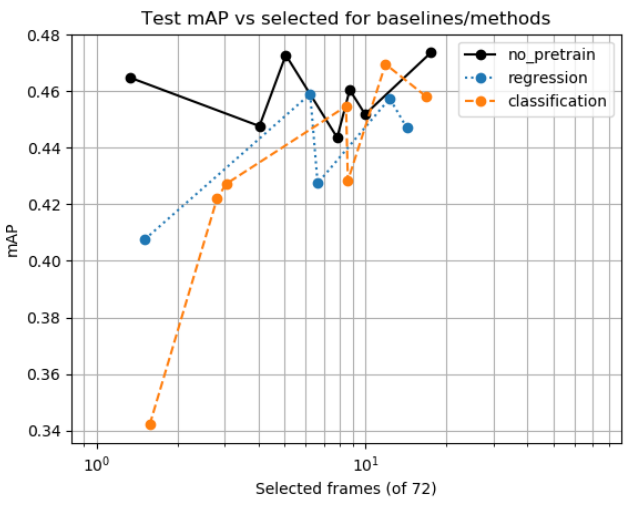


(a) for GTEA Gaze+

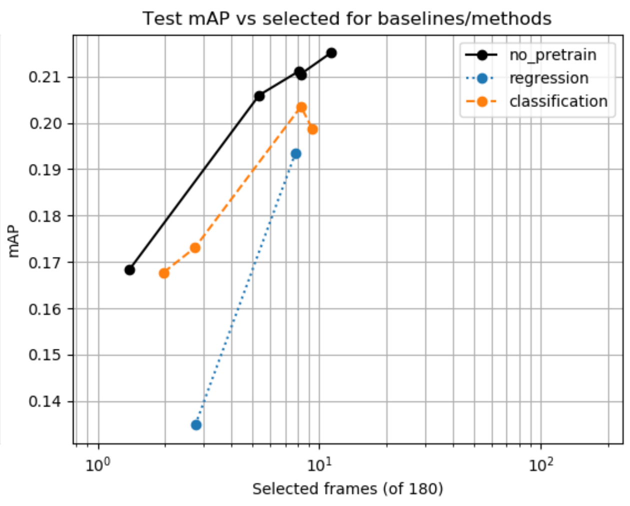


(b) for EPIC-KITCHENS

Figure 11: mAP versus frames selected for datasets - joint finetuning. The effects of joint finetuning seem to vary; for GTEA Gaze+, the performance increases for many cases, while for EPIC-KITCHENS the performance decreases slightly.



(a) for GTEA Gaze+



(b) for EPIC-KITCHENS

Figure 12: mAP versus frames selected for datasets - volume pretraining. Both classification and regression pretraining decrease the performance of the model, though regression decreases it more.

4.6.2 Volume Pretraining

For this variant, we added a Volume Pretraining stage before the FrameSelector-only stage. Recall there are two types of pretraining: one where we treat matching the volume as a regression problem, and one as a binary classification problen. We show the effects of adding this stage in Figure 12.

For both datasets, introducing a pretraining stage causes the performance to decrease. Notably, the regression variant causes a more significant decrease than the classification variant. This difference could be attributed to the scaling of the learning rates: the regression variant used mean squared error loss, while the classification variant used cross-entropy. One possible reason that pretraining negatively affects the performance is that pretraining causes the FrameSelector to overfit to the volume. This causes the weights of the FrameSelector to not learn as much during the FS-only stage proper, which results in worse selections.

4.6.3 Summary

In summary, of the variants we tried, none noticeably improve the performance of our model, and many even reduce it significantly. Thus, when training the model, it is best to use only the FC-only and FS-only stages.

5 Conclusion and Future Work

In conclusion, we have found that our method as proposed outperforms several baselines, including naive baselines based on audio. In fact, for one dataset, our model performs as well as a state-of-the-art model [38]. However, our model did not significantly improve on some baselines we considered. One likely component was the significant amount of per-epoch variation for performance, which made it difficult to deterministically compare baselines and methods. Alternatively, it might be that the optimal frame selection algorithm, when only provided with a spectrogram, is in fact to just output a near-constant policy.

There are many avenues to continue this research, which could potentially improve the performance or computation time.

One potential improvement is to improve the FrameClassifier model. For example, the averaging layer could be replaced with a long short-term memory (LSTM) layer that processes all of the selected frames in sequence. This preserves temporal information, which could be useful when the order of frames is important to classifying an action, such as “open” vs “close”.

There are also many possibilities of changing the FrameSelector model. Currently the FrameSelector treats the entire spectrogram as a whole to output the selection. Instead, one could have components of the FrameSelector’s feature representation correspond to different temporal locations (left and right along the spectrogram). While we considered this at first, it is not clear how upsampling the output of the FrameSelector to correspond to frames would work in this case. One could also consider different architectures entirely. For example, the FrameSelector could be replaced with a variation of a segmentation network, where the output tensor has the same height and width as the input.

The current model looks at the spectrogram exactly once, then looks at the selected subset of frames. Another method, more similar to AdaFrame [36], would be to let the model look “back-and-forth” between the frames and the spectrogram. An example of where this might be useful is a clip with two seemingly useful patterns on a spectrogram, where the first one appears more prominently but is actually background noise. Using our current method, our model might select frames both patterns, resulting in a high computation time, or only select frames from the first, resulting in a low accuracy. The alternate model would be able to look at very few frames of the first type, and upon realizing the frames are not useful, ignore the pattern wherever it appears in the spectrogram.

6 Bibliography

[1] Mehmet Ali Arabacı, Fatih Özkan, Elif Surer, Peter Jančovič, and Alptekin Temizel. Multi-modal egocentric activity recognition using audio-visual features. *arXiv preprint arXiv:1807.00612*, 2018.

[2] Tadas Baltrušaitis, Chaitanya Ahuja, and Louis-Philippe Morency. Multimodal machine learning: A survey and taxonomy. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 41(2):423–443, 2019.

[3] 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.

[4] Dima Damen, Hazel Doughty, Giovanni Maria Farinella, Sanja Fidler, Antonino Furnari, Evangelos Kazakos, Davide Moltisanti, Jonathan Munro, Toby Perrett, Will Price, and Michael Wray. Scaling egocentric vision: The epic-kitchens dataset. In *European Conference on Computer Vision (ECCV)*, 2018.

[5] James W Davis and Ambrish Tyagi. Minimal-latency human action recognition using reliable-inference. *Image and Vision Computing*, 24(5):455–472, 2006.

[6] Marc Peter Deisenroth, Gerhard Neumann, Jan Peters, et al. A survey on policy search for robotics. *Foundations and Trends® in Robotics*, 2(1–2):1–142, 2013.

[7] Hehe Fan, Zhongwen Xu, Linchao Zhu, Chenggang Yan, Jianjun Ge, and Yi Yang. Watching a small portion could be as good as watching all: Towards efficient video classification. In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI-18*, pages 705–711. International Joint Conferences on Artificial Intelligence Organization, 7 2018.

[8] Alireza Fathi, Yin Li, and James M Rehg. Learning to recognize daily actions using gaze. In *European Conference on Computer Vision*, pages 314–327. Springer, 2012.

[9] Michael Figurnov, Maxwell D Collins, Yukun Zhu, Li Zhang, Jonathan Huang, Dmitry Vetrov, and Ruslan Salakhutdinov. Spatially adaptive computation time for residual networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1039–1048, 2017.

[10] Alex Graves. Adaptive computation time for recurrent neural networks. *arXiv preprint arXiv:1603.08983*, 2016.

[11] Samitha Herath, Mehrtash Harandi, and Fatih Porikli. Going deeper into action recognition: A survey. *Image and vision computing*, 60:4–21, 2017.

[12] Minh Hoai and Fernando De la Torre. Max-margin early event detectors. *International Journal of Computer Vision*, 107(2):191–202, 2014.

[13] Gao Huang, Danlu Chen, Tianhong Li, Felix Wu, Laurens van der Maaten, and Kilian Q Weinberger. Multi-scale dense networks for resource efficient image classification. *arXiv preprint arXiv:1703.09844*, 2017.

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

[15] Sourabh Kulhare, Shagan Sah, Suhas Pillai, and Raymond Ptucha. Key frame extraction for salient activity recognition. In *2016 23rd International Conference on Pattern Recognition (ICPR)*, pages 835–840. IEEE, 2016.

[16] Li Liu, Ling Shao, and Peter Rockett. Boosted key-frame selection and correlated pyramidal motion-feature representation for human action recognition. *Pattern recognition*, 46(7):1810–1818, 2013.

[17] Mason McGill and Pietro Perona. Deciding how to decide: Dynamic routing in artificial neural networks. In *International Conference on Machine Learning*, pages 2363–2372, 2017.

[18] Youssef Mroueh, Etienne Marcheret, and Vaibhava Goel. Deep multimodal learning for audio-visual speech recognition. In *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 2130–2134. IEEE, 2015.

[19] Jiquan Ngiam, Aditya Khosla, Mingyu Kim, Juhan Nam, Honglak Lee, and Andrew Y Ng. Multimodal deep learning. In *Proceedings of the 28th international conference on machine learning (ICML-11)*, pages 689–696, 2011.

[20] Bowen Pan, Wuwei Lin, Xiaolin Fang, Chaoqin Huang, Bolei Zhou, and Cewu Lu. Recurrent residual module for fast inference in videos. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1536–1545, 2018.

[21] Steven J Rennie, Etienne Marcheret, Youssef Mroueh, Jerret Ross, and Vaibhava Goel. Self-critical sequence training for image captioning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 7008–7024, 2017.

[22] Michael S Ryoo. Human activity prediction: Early recognition of ongoing activities from streaming videos. In *2011 International Conference on Computer Vision*, pages 1036–1043. IEEE, 2011.

[23] 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.

[24] Yu-Chuan Su and Kristen Grauman. Leaving some stones unturned: dynamic feature prioritization for activity detection in streaming video. In *European Conference on Computer Vision*, pages 783–800. Springer, 2016.

[25] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction.* MIT press, 2018.

[26] Naoya Takahashi, Michael Gygli, and Luc Van Gool. Aenet: Learning deep audio features for video analysis. *IEEE Transactions on Multimedia*, 20(3):513–524, 2018.

[27] Surat Teerapittayanon, Bradley McDanel, and HT Kung. Branchynet: Fast inference via early exiting from deep neural networks. In *2016 23rd International Conference on Pattern Recognition (ICPR)*, pages 2464–2469. IEEE, 2016.

[28] 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.

[29] Andreas Veit and Serge Belongie. Convolutional networks with adaptive inference graphs. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 3–18, 2018.

[30] Paul Viola and Michael J Jones. Robust real-time face detection. *International journal of computer vision*, 57(2):137–154, 2004.

[31] 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.

[32] Xiaolong Wang, Ross Girshick, Abhinav Gupta, and Kaiming He. Non-local neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 7794–7803, 2018.

[33] Xin Wang, Fisher Yu, Zi-Yi Dou, Trevor Darrell, and Joseph E Gonzalez. Skipnet: Learning dynamic routing in convolutional networks. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 409–424, 2018.

[34] Qiuxia Wu, Zhiyong Wang, Feiqi Deng, and David Dagan Feng. Realistic human action recognition with audio context. In *2010 International Conference on Digital Image Computing: Techniques and Applications*, pages 288–293. IEEE, 2010.

[35] Zuxuan Wu, Tushar Nagarajan, Abhishek Kumar, Steven Rennie, Larry S Davis, Kristen Grauman, and Rogerio Feris. Blockdrop: Dynamic inference paths in residual networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 8817–8826, 2018.

[36] Zuxuan Wu, Caiming Xiong, Chih-Yao Ma, Richard Socher, and Larry S Davis. Adaframe: Adaptive frame selection for fast video recognition. *arXiv preprint arXiv:1811.12432*, 2018.

[37] Saining Xie, Chen Sun, Jonathan Huang, Zhuowen Tu, and Kevin Murphy. Rethinking spatiotemporal feature learning: Speed-accuracy trade-offs in video classification. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 305–321, 2018.

[38] Serena Yeung, Olga Russakovsky, Greg Mori, and Li Fei-Fei. End-to-end learning of action detection from frame glimpses in videos. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2678–2687, 2016.

[39] Bowen Zhang, Limin Wang, Zhe Wang, Yu Qiao, and Hanli Wang. Real-time action recognition with enhanced motion vector cnns. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2718–2726, 2016.

[40] Zhipeng. Zhao and Ahmed M Elgammal. Information theoretic key frame selection for action recognition. In *Proc. BMVC*, pages 109.1–109.10, 2008.

[41] Xizhou Zhu, Yuwen Xiong, Jifeng Dai, Lu Yuan, and Yichen Wei. Deep feature flow for video recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2349–2358, 2017.