Match Cutting: Finding Cuts with Smooth Visual Transitions

Boris Chen Amir Ziai Rebecca S. Tucker Yuchen Xie
{bchen, aziai, btucker, yxie}@netflix.com
Netflix Inc.
Los Gatos, CA, USA

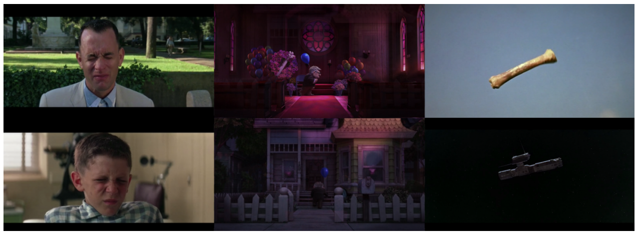

Figure 1. Three example match cuts where the framing of the subject is matched: (left) Forrest Gump (1994), (center) Up (2009), and (right) 2001: A Space Odyssey (1968).

Abstract

A match cut is a transition between a pair of shots that uses similar framing, composition, or action to fluidly bring the viewer from one scene to the next. Match cuts are frequently used in film, television, and advertising. However, finding shots that work together is a highly manual and time-consuming process that can take days. We propose a modular and flexible system to efficiently find high-quality match cut candidates starting from millions of shot pairs. We annotate and release a dataset of approximately 20k labeled pairs that we use to evaluate our system, using both classification and metric learning approaches that leverage a variety of image, video, audio, and audio-visual feature extractors. In addition, we release code and embeddings for reproducing our experiments at github.com/netflix/matchcut.

1. Introduction

In film, a shot is a series of frames representing an uninterrupted period of time between two cuts [12]. A match cut is a transition between a pair of shots that uses similar framing, composition, or action to fluidly bring the viewer from one scene to the next. It is a powerful visual storytelling tool used to create a connection between two scenes.

For example, a match cut from a person to their younger or older self is commonly used in film to signify a flashback or flash-forward to help build the backstory of a character. Two example films that used this are Forrest Gump (1994) [79] and Up (2009) [21] (Fig. 1). Without this technique, a narrator or character might have to explicitly verbalize that information, which may ruin the flow of the film.

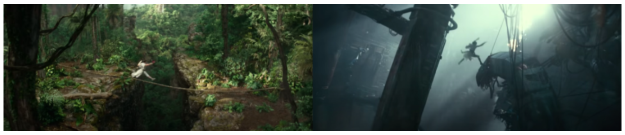

Figure 2. A match cut from the *Star Wars: The Rise of Skywalker* (2019) [1] trailer. The trailer editor took two shots from the different scenes with similar jump motions and cut them together. The matched motion gives the illusion of one continuous jump.

A famous example from Stanley Kubrik's 2001: A Space Odyssey [43] is also shown in Fig. 1. This iconic match cut from a spinning bone to a spaceship instantaneously takes the viewer forward millions of years into the future. It is a highly artistic edit which suggests that mankind's evolution from primates to space technology is natural and inevitable.

Match cuts can use any combination of elements, such as framing, motion, action, subject matter, audio, lighting, and color. In this paper, we will specifically address two types: (1) character frame match cuts, in which the framing of the character in the first shot aligns with the character in the second shot, and (2) motion match cuts, where shots are matched together on the basis of general movement. Motion match cuts can use common camera movement (pan left/right, zoom in/out) or motion of subjects. They create the feeling of smooth transitions between inherently discontinuous shots. An example is shown in Fig. 2.

Match cutting is considered one of the most difficult video editing techniques [22], because finding a pair of shots that match well is tedious and time-consuming. For a feature film, there are approximately 2k shots on average, which translates to 2M possible shot pairs, the vast majority of which will not be good match cuts. An editor typically watches one or more long-form videos and relies on memory or manual tagging to identify shots that would match to a reference shot observed earlier. Given the large number of shot pairs that need to be compared, it is easy to overlook many desirable match cuts.

Our goal is to make finding match cuts vastly more efficient by presenting a ranked list of match cut pair candidates to the editors, so they are selecting from, e.g., the top 50 shot pairs most likely to be good match cuts, rather than millions of random ones. This is a challenging video editing task that requires complex understanding of visual composition, motion, action, and sound.

Our contributions in this paper are the following: (1) We propose a modular and flexible system for generating match cut candidates. Our system has been successfully utilized by editors in creating promotional media assets (e.g. trailers) and can also be used in post-production to find matched shots in large amount of pre-final video. (2) We release a dataset of roughly 20k labeled match cut pairs for two types of match cuts: character framing and motion. (3) We evaluate our system using classification and metric learning approaches that leverage a variety of image, video, audio, and audio-visual feature extractors. (4) We release code and embeddings for reproducing our experiments.

2. Related Work

Computational video editing There is no computational or algorithmic approach to video editing that matches the skill and creative vision of a professional editor. However, a number of methods and techniques have been proposed to address sub-problems within video editing, particularly the automation of slow and manual tasks.

Automated video editing techniques for specialized non-fiction videos has seen success with rules-based methods, such as those for group meetings [58, 64], educational lectures [31], interviews [8] and social gatherings [5]. Broadly speaking, these methods combine general film editing conventions (e.g. the speaker should be shown on camera) with heuristics specific to the subject domain (e.g. for educational lectures, the white board should be visible).

Computational video editing for fictional works tends to fall in one of two lines of research: transcript-based approaches [45, 72, 25, 68] and learning-based approaches [53]. Leake et. al. [45] generates edited video sequences using a standard film script and multiple takes of the scene, but their work is specific to dialogue-driven scenes. Two similar concepts, Write-A-Video [72] and QuickCut [68], generate video montages using a combination of text and a video library. Learning-based approaches have seen success in recent years, notably in Learning to Cut [53], which proposes a method to rank realistic cuts via contrastive learning [20]. The MovieCuts dataset [54] includes match cuts as a subtype, though it is by far the smallest category and does not distinguish between kinds of match cuts. In contrast, we release a data set of 20k pairs that differentiate between frame and motion cuts, with the goal of finding these pairs from shots throughout the film instead of detecting existing cuts. Our work advances learning-based computational video editing by introducing a method to generate and then rank proposed pairs of match cuts without fixed rules or transcripts.

Video Representation Learning Self-supervised methods have dominated much of the progress in multi-modal media understanding in recent years [76, 47, 24, 37]. CLIP [57] was an early example of achieving impressive zero-shot visual classification following self-supervised training with over 400M image-caption pairs. Similar advances have been made for audio [29] and video [50] by utilizing different augmented views of the same modality [19, 18, 27, 60], or by learning joint embeddings of short [29, 3, 50] or long-form [39] videos. Our system leverages such work for learning video representations that capture matching video pairs for the task of match cutting.

Movie understanding There is a deep and rich literature on models that understand and analyze the information in movies. Many movie-specific datasets [34] have been developed that have enabled research into a variety of topics such as human-centric situations [70], story-based retrieval [6], shot type classification [59], narrative understanding [6, 10, 44], and trailer analysis [35]. We release a dataset which contributes a novel and challenging movie understanding task.

3. Methodology

In this section, we present a flexible and scalable system for finding K matching shot pairs given a video. This system consists of five steps, as depicted in Fig. 3.

3.1. Preprocessing

The first two steps of our system segment a video into a sequence of contiguous and non-overlapping shots and remove near-duplicate shots. Although we present concrete implementations for these steps, our system is agnostic to these choices.

Step 1: Shot segmentation. For each movie m, we run a shot segmentation algorithm to split that title into n_m shots. Let $S^m = \{s_i^m\}_{i=1}^{n_m}$ be the set of shots where s_i^m corresponds to the i-th shot of the m-th movie. Shot s_i^m consists of an ordered set of frames $F_i^m = \{f_{(i,j)}^m\}_{j=1}^{l_i^m}$, where $f_{(i,j)}^m$ is the j-th frame of s_i^m, and l_i^m is the number of frames in s_i^m. We use a custom shot segmentation algorithm but similar results can by achieved with PySceneDetect [15] or TransNetV2 [62].

Step 2: Near-duplicate shot deduplication. Matching shots should have at least one difference in character, background, clothing, or character age. Therefore, we remove near-duplicate shots (e.g. two shots of the same character in the same scene and framing, but with a slightly different facial expression).

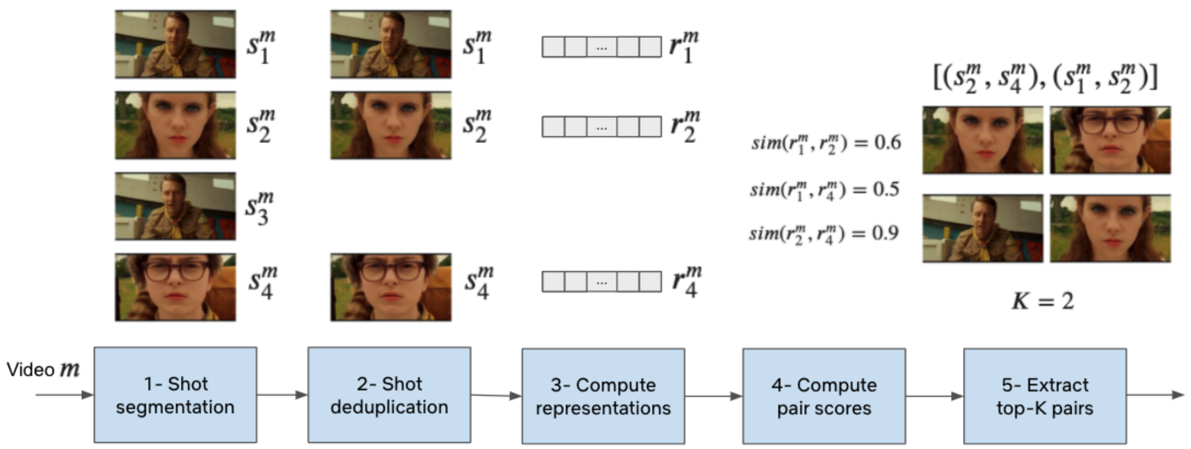

Figure 3. System diagram for generating candidate match cut pairs. The input is a video file for movie m and the output is K match cut candidates. (1) Video is split into shots using a shot segmentation algorithm. (2) Near-duplicate shots are removed. (3) A tensor representation r_i^m is computed for each shot s_i^m using an encoder. (4) All unique shot pairs are enumerated and a score function sim is used to compute the similarity between shot representations. (5) The top-K pairs with highest similarity are returned. We show an illustrative example with four shots from Moonrise Kingdom (2012) [4] and $K = 2$.

Our specific methodology for deduplication is as follows: we first extract the center frame c_i^m for each shot s_i^m defined as $c_i^m = f^m_{(i, \lfloor l_i^m/2 \rfloor)}$. For each center frame, we extract the penultimate embeddings out of MobileNet [33] pretrained on ImageNet [42]. Let $e_i^m = \text{enc}(c_i^m) \in \mathbb{R}^{1024}$ be the embedding for frame c_i^m where enc takes an image and outputs a 1024-dimensional vector.

We define the set of duplicate shot indices for movie m as

$$D^m = \{j | i, j \in \{1, 2, \ldots, n_m\}, i < j, \cos(e_i^m, e_j^m) \geq T_d\} \tag{1}$$

where cos computes the cosine similarity between a pair of embeddings and T_d is the similarity threshold.

Finally, the set of deduplicated shots for movie m can be constructed by excluding the shots corresponding to the indices in D^m as follows: $S_d^m = \{s_i^m | i \in \{1, 2, \ldots, n_m\}, i \notin D^m\}$. We leverage the `imagededup` [36] library and find that setting $T_d = 0.8$ removes most of the near-duplicates.

3.2. Shot Pair Ranking

Steps 3-5 score and rank pairs of deduplicated shots following step 2.

Step 3: Shot representation computation. In this step, we compute a tensor representation r_i^m for each shot s_i^m. Representations for different shots need to preserve some notion of similarity for matching pairs. Representations can be extracted using any video, image, audio, text, or multi-modal encoders. We present a few such choices in the upcoming sections.

Step 4: Shot pair score computation. In this step, we enumerate all unique shot pairs for movie m,

$$P^m = \{(s_i^m, s_j^m) | s_i^m, s_j^m \in S_d^m, i < j\} \tag{2}$$

and compute a similarity score $\text{sim}(r_i^m, r_j^m) \in \mathbb{R}$ for each pair of shots (s_i^m, s_j^m). This similarity score is used for ranking pairs where higher-scoring pairs are considered higher quality. The function sim can be any function that takes a pair of tensors and outputs a real scalar. This function can be chosen beforehand (e.g. cosine similarity) or learned through supervision.

Step 5: Top-K pair extraction. This step simply ranks the results from the previous step and returns the top-K pairs.

3.3. Heuristics

We define a heuristic h as a specific combination of shot representation and predetermined scoring function. These heuristics serve two functions. We use them to *generate* candidate pairs for manual annotation by video editors, and then also to *evaluate* the annotated data set. Here, evaluate means that we use the heuristic to rank the candidate pairs and compute the average precision of that ranked list. More details about evaluation can be found in Supplementary 3.

We leverage four of the heuristics presented in this section (h_1, h_2, h_4, and h_5) to generate candidate pairs for annotation in Sec. 4 and report how all of the heuristics perform on our dataset in Sec. 5.

Heuristic 1 (h_1): equal number of faces. One very crude heuristic for character frame match cutting is to consider pairs where the number of faces between the two shots is equal. For the shot representation, we extract the center frame and use a face detection model (Inception-ResNet-v1 [63] pretrained on VGGFace2 [13]) to determine the number of faces. The scoring function outputs 1 if the two shots have the same number of faces, and 0 otherwise.

Heuristics 2 (h_2) and 3 (h_3): Instance segmentation.

These heuristics are designed for character frame matching. We leverage instance segmentation to extract pixel-level representations of the presence of people in a frame. In other words, we can extract the silhouette of characters, which contain rich information about the character's framing in the image.

The instance segmentation model takes a single center frame c_i^m and returns a set of instance-level binary masks $B_i^m = \{b_{(i,x)}^m\}_{x=1}^{u_i^m}$, where u_i^m is the number of such instances, $b_{(i,x)}^m \in \{0,1\}^{W \times H}$ is the binary mask of the x-th instance, W is the width of the binary mask, and H is the height (same shape as c_i^m).

We use the union of all binary masks $b_i^m = \bigcup_{x=1}^{u_i^m} b_{(i,x)}^m$ as the shot representation r_i^m for heuristic 2 (h_2). Intuitively, a perfect character match cut will involve characters with the exact same binary mask. The Intersection over Union (IoU) metric captures this notion well and we use it as the similarity score for h_2. Concretely, IoU takes two binary masks and returns a real value between 0 and 1, which captures how well the masks overlap (the higher the better):

$$IoU(b_i^m, b_j^m) = |b_i^m \cap b_j^m| / |b_i^m \cup b_j^m| \tag{3}$$

where b_i^m, b_j^m are binary masks for center frame c_i^m and c_j^m, $b_i^m \cap b_j^m$ is the set of person pixels that are shared between the two masks, and $b_i^m \cup b_j^m$ is the set of pixels that contain a person pixel in at least one of the masks. If either mask is empty, we set $IoU = 0$.

Taking the union of instance-level binary masks can lead to matching the wrong number of characters between two shots, as depicted in Fig. 4. Instead, for heuristic 3 (h_3) we use B_i^m as the representation of s_i^m and use Instance IoU as the similarity score function.

We start by associating character instances across center frames, which ensures that each instance in B_i^m is matched to at most one instance in B_j^m. Inspired by SORT [9], we formulate this association as a linear assignment problem with assignment cost computed using the negative IoU of instance masks. We denote $A_{(i,j)}^m$ as the set of associated instance mask pairs between c_i^m and c_j^m. The instance-level IoU (IIoU) is computed as follows:

$$IIoU(B_i^m, B_j^m) = \frac{\sum_{(b_{(i,x)}^m, b_{(j,y)}^m) \in A_{(i,j)}^m} |b_{(i,x)}^m \cap b_{(j,y)}^m|}{|b_i^m \cup b_j^m|} \tag{4}$$

Even with this improvement, both of these heuristics fail in at least two scenarios. First, segmentation inaccuracies can lead to false positives or false negatives. Second, two frames with very different character poses sometimes produce highly similar binary masks (e.g. matching the face of one character to the back of another).

We use the PyTorch [55] implementation of Mask R-CNN with a ResNet-50-FPN backbone [30, 75], pretrained on COCO train2017 [46], filter out all instance types except for "person", and use a 0.5 threshold.

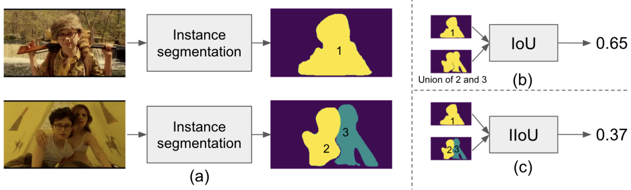

Figure 4. (a) Two frames from the Moonrise Kingdom [4] are passed through an instance segmentation network to obtain instance-level binary masks. (b) For h_2 we use the union of masks as the representation and IoU as the similarity score function. (c) for h_3 we preserve the instance-level binary masks for the representations, and use IIoU as the similarity score function.

Heuristics 4 (h_4) and 5 (h_5): Optical Flow. We use heuristics 4 and 5 to generate annotation candidates for motion match cutting. The editors are looking for pairs of shots with similar motions so that they can edit together to create the continuation. The criterion for good motion match cuts is a similarity in the movement of the camera or subjects between the shots. Because movement is so critical to this kind of match cut, we cannot simply take the static center frame of each shot as we do for heuristics 1-3.

Instead, we need a way to quantify the motion of the shot across all frames. Optical flow refers to the task of estimating the movement of boundaries, edges, and objects within a video or sequence of ordered images. Dense optical flow is the task of estimating the movement of each pixel within a video frame. Sparse optical flow, on the other hand, only tracks the movement of key points within the frame. Optical flow has been traditionally formulated as an optimization problem [32, 16, 78] or as a gradient-based estimation. However, in recent years, deep-learning based models have become a viable alternative to these methods.

Both heuristics for motion match cuts use the following general procedure. For each sequential pair of frames in the shot s_i^m, we compute the optical flow, which yields a tensor of size $W \times H \times 2$, representing the horizontal and vertical motion for each pixel. Here W and H are the width and height, respectively, of the frame in pixels. The optical flow tensor for consecutive frames $f_{(i,j)}^m$ and $f_{(i,j+1)}^m$ is $q_{(i,j)}^m = OF(f_{(i,j)}^m, f_{(i,j+1)}^m)$, where OF is the specific choice of optical flow implementation.

For each shot, we average the optical flow tensors across all the frames in the shot s_i^m:

$$Q_i^m = \left(\sum_{j=1}^{l_i^m - 1} q_{(i,j)}^m \right) / (l_i^m - 1) \tag{5}$$

This allows us to compare shots of different length. An example optical flow output can be seen in Fig. 5.

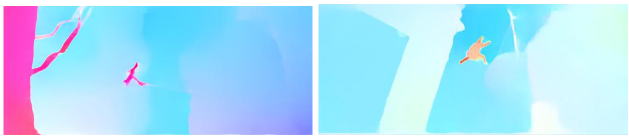

Figure 5. An optical flow result for the match cut in Fig. 2 from the trailer for *Stars Wars: The Rise of Skywalker* [1]. In this visualization, the color represents the direction of motion. Pink pixels are moving to the right and blue pixels are moving to the left. The intensity of the color represents the magnitude of the motion.

For heuristic 4 (h_4), we use the `opencv` implementation [11] of the Farnebäck method to compute dense optical flow [23]. This method approximates the neighborhood of each pixel with a quadratic polynomial and uses differences in the polynomials to estimate the per-pixel displacement.

For heuristic 5 (h_5), we use a deep-learning-based method: Recurrent All-Pairs Field Transforms (RAFT) [66]. Its network architecture comprises three components: a feature extractor, a correlation layer that creates a 4D correlation volume for all pixel pairs, and a recurrent GRU-based update operator. We are using the version of the model that has been pretrained on the FlyingThings [51] dataset. The optical flow visualizations in Fig. 5 were generated using RAFT.

We use $r_i^m = flatten(Q_i^m)$ and cosine similarity as the similarity function for this heuristic. In practice, we sample every four frames to save computation and achieve similar results to using every frame.

3.4. Scalability

Our proposed system can be used to find match cuts both within a title and across multiple titles. However, the number of shot pairs that need to be compared is quadratic in the number of shots, which can quickly become a bottleneck if we want to find match cuts across multiple titles. To avoid this, we can replace steps 4 and 5 in Section 3.2 with an approximate nearest neighbors approach (ANN) such as FAISS [38]. For this approach, we first need to build an index of the representations that we computed in Section 3.2. Once this index is built, we can retrieve the top-K results for each shot, and then compute the global top-K pairs.

Although this approach is significantly more efficient, it presents three issues. First, the approximate nature of ANN methods may lead to imprecise results. Second, ANN methods don't support arbitrary functions for computing nearest neighbors. Third, representations must have the same shape. We explore this trade-off further in Sec. 5.

4. Data

The dataset we release with this paper contains ∼20k pairs of labeled shots. We include seven embeddings for each shot (described in Sec. 5), shot boundary timestamps, and the annotated labels. In this section, we describe the process for the collection of this dataset.

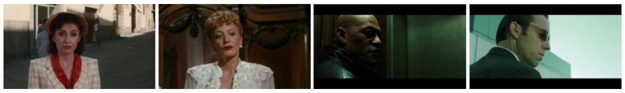

Figure 6. (left) A frame match cut from *Life is Beautiful* (1997) [7], (right) A frame match cut from *The Matrix* (1999) [71].

4.1. Movie Set Selection and Pre-processing

Here we select a set of movies, segment these movies into shots, remove near-duplicate shots, and construct the set of all shot pairs within the same movie (intra-movie).

Movie set selection We selected 100 movies from the MovieNet [34] dataset which are diversified across genre, release year, and country of origin. The full list of movies can be found in Supplementary 2.

Shot segmentation We segmented each of the 100 movies into shots as described in Sec. 3.1. Following this step, we are left with 128k shots across all movies, which translates into over 8.2B unique shot pairs that could be considered for annotation.

Shot deduplication We use the deduplication methodology described in Sec. 3.1 to remove near-duplicate shots. This step shrinks the overall number of shots by over 40%, from 128k to 75k, which also shrinks the number of pairs to 2.8B.

Limit to intra-movie matches Though inter-movie match cuts are also interesting, for this study we only consider intra-movie pairs–which leaves ∼35M shot pairs. The vast majority of these pairs are not match cuts, so we employ heuristics to further reduce the annotation candidate set and increase the likelihood of finding positive pairs relative to random sampling.

4.2. Annotation Candidate Pair Generation

Using the heuristics discussed in Sec. 3.3, we score, rank, and retrieve the top 50 shot pairs for each movie m, which we denote $P_{h_i}^m$. Note that $P_{h_i}^m \subseteq P^m$ and that $|P_{h_i}^m| = 50$. For each type we utilized two heuristics h_i and h_j, and used the union of the resultant sets as the dataset for annotation.

We use h_1 and h_2 for character frame match cutting and h_4 and h_5 for motion match cutting. (Heuristic h_3 was not available during the annotation process.) In other words, $P_{h_1}^m \cup P_{h_2}^m$ is the set of pairs that we annotated for character frame, and $P_{h_4}^m \cup P_{h_5}^m$ for motion.

Heuristic-discovered match cuts We present a few examples from our match cutting heuristics. Fig. 6 contains examples of character frame match cut h_2. Fig. 7 is an example of a motion match cut found by h_5.

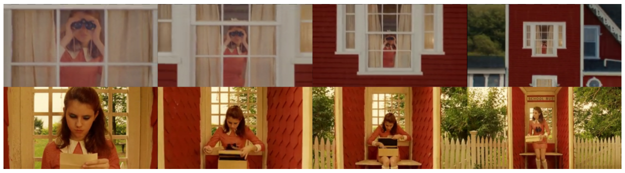

Figure 7. A motion match cut from *Moonrise Kingdom* (2012) [4] where the primary motion detected by optical flow is from the camera zooming out.

4.3. Data Collection

4.3.1 Task Definitions

Our data collection process started with inspecting and refining the definitions of each match cutting type with a group of three senior in-house video editors. After arriving at consistent definitions, we developed reference material, which included illustrative examples of positive and negative pairs.

Our senior video editors trained two separate sets of three annotators, one for each type. Due to the technical nature of this task, we selected video editors at a video editing agency that we had previously worked with. We asked annotators to assign a binary label to each pair they were presented with. The training involved an hour-long walkthrough of guidelines and examples.

4.3.2 Annotations

We used six annotators: three for character frame match cuts and three for motion match cuts, so each candidate pair was annotated by three annotators. For frame match cuts, our annotators labeled 9,985 pairs and were in perfect agreement (all 3 annotators chose the same label) for 84% of the pairs. For motion match cuts, our annotators classified 9,320 pairs and were in perfect agreement for 75% of them. We attribute the lower annotator agreement for motion match cutting to the fact that it is a more difficult, subjective, and nuanced task. We use the majority vote for each pair as the final label for the rest of this paper. However, annotator-level data is also made available in the dataset. Roughly 8.7% of frame match pairs were majority labeled positive, and 9.9% for motion.

4.3.3 Random negative pairs

Since we used heuristics to generate annotation candidates, our dataset may not reflect many pairs that the models are likely to encounter. For instance, we only used shots with faces of people for frame match cutting, but our system needs to be able to score pairs that contain no faces. To reduce this bias, we randomly sample 50 pairs for each title and append these pairs as negative examples to the dataset we use for training and evaluating models in Sec. 5. These pairs have no overlap with the annotated pairs and, since they were drawn at random, they are extremely unlikely to contain positive pairs. More information about the dataset and its associated statistics can be found in Supp. 1.

5. Experiments

Our experiments explore two goals: finding a scalable solution to a quadratic problem and high retrieval quality. The heuristic-based candidate generators are useful to aid labeling by suggesting pairs for annotation, but a learned model can perform better than heuristics. High model accuracy translates to saved time and higher quality cuts for our video editors.

Experiment 1 explores a binary classification setup, which gives us high retrieval quality, but at the expense of scalability, since every shot pair will need to be run through the classifier. Experiment 2 uses metric learning to provide both high retrieval quality and scalability. For both sets of experiments, the shot segmentation and deduplication steps remain fixed.

5.1. Setup

We split our dataset at the movie level by randomly selecting 60, 20, and 20 titles for train, validation, and test sets respectively. For all experiments we train 5 models using different seeds, compute average precision (AP) on the validation dataset (AP_{val}) for each model, and finally report test AP (AP_{test}) for the model with the highest AP_{val} in addition to the mean and standard deviation of AP_{val} across the 5 runs.

5.2. Experiment 1: Binary classification

In this experiment, we explore pretrained image, video, and audio networks as embedding (i.e. representation) extractors. Once we extract penulatimate layer embeddings for each shot, we train a binary classifier, which we use to report evaluation metrics. Concretely, for each pair of shots s_i^m and s_j^m, we extract representations r_i^m and r_j^m. These tensors are then aggregated using a function that takes two tensors and outputs a single tensor. The aggregated tensor is used as the input into the binary classification model. We experimented with using one layer prior to the penultimate layer as well as end-to-end training and were not able to produce reasonable results. We hypothesize that this could be due to the relatively small size of our labeled data.

We experimented with 5 binary classifiers: (1) XGBoost (XGB) [17] with 100 boosted rounds, unbounded depth, and logistic objective, (2) logistic regression (LR) with L_2 penalty, $C = 1$, and LBFGS solver, (3) 2-layer multi-layer perceptron (MLP$_S$) with ReLU activation [2], Adam optimizer [41], lr=0.001, 200 max epochs, and 50 units in each hidden layer, (4) MLP$_M$ same as (3) but with 100 units in each layer, and (5) MLP$_L$ same as (3) but with 500 units.

Table 1. Experiment 1: Binary Classification Results.

	Character frame				Motion			
method	model	agg	AP_{val}	AP_{test}	model	agg	AP_{val}	AP_{test}
random	-	-	0.094	0.119	-	-	0.096	0.122
h_1	-	-	0.006	0.017	-	-	-	-
h_2	-	-	0.177	0.207	-	-	-	-
h_3	-	-	0.234	**0.248**	-	-	-	-
h_4	-	-	-	-	-	-	0.192	**0.163**
h_5	-	-	-	-	-	-	0.134	0.132
CLIP	MLP$_M$	mean	0.253±0.023	0.240	MLP$_L$	cat	0.131±0.010	0.107
RN50	MLP$_L$	cat	0.266±0.015	0.269	MLP$_M$	mean	0.120±0.010	0.136
EN7	XGB	mean	0.261±0.025	**0.352**	MLP$_L$	mean	0.118±0.007	0.129
R(2+1)D	MLP$_L$	mean	0.222±0.021	0.224	MLP$_L$	cat	0.184±0.022	**0.193**
Swin	MLP$_M$	mean	0.270±0.030	0.287	MLP$_S$	mean	0.155±0.016	0.150
C4C	MLP$_L$	mean	0.277±0.012	0.304	MLP$_M$	mean	0.121±0.007	0.130
YN	XGB	cat	0.174±0.015	0.160	MLP$_L$	diff	0.136±0.016	0.136
YN-CLIP	XGB	cat	0.245±0.022	0.286	XGB	diff	0.137±0.016	0.137
YN-RN50	XGB	cat	0.283±0.015	0.321	XGB	mean	0.129±0.007	0.145
YN-EN7	XGB	mean	0.275±0.030	**0.355**	XGB	diff	0.136±0.011	0.128
YN-R(2+1)D	XGB	cat	0.219±0.020	0.218	XGB	mean	0.170±0.015	**0.177**
YN-Swin	MLP$_L$	mean	0.269±0.024	0.336	XGB	mean	0.139±0.010	0.161
YN-C4C	XGB	mean	0.289±0.017	0.327	XGB	mean	0.140±0.008	0.124

We use the XGBoost [17] library for (1) and scikit-learn [56] for (2-5). We also explored 3 aggregation functions (agg): concatenation (cat), mean pooling (mean), and pairwise absolute distance (diff).

For each encoder and type we train all 15 combinations of models and aggregation functions as described earlier. We then report the combination with the highest mean AP_{val}.

5.2.1 Encoders

We used a total of seven image, video, and audio encoders. For image encoders, we use CLIP [57], ResNet-50 (RN50) [30], and EfficientNet-B7 (EN7) [65]. For RN50 and EN7 we use the PyTorch [55] implementation pretrained on ImageNet [42]. We pass the center frame through the network and extract the penultimate layer embeddings.

For video encoders, we use Video Swin [49], R(2+1)D [67], and CLIP4Clip (C4C) [50]. For Video Swin, we uniformly sample four views of 32 frames with stride 2 in the temporal dimension. For R(2+1)D, we sample every four frames as described in Sec. 3.3. For C4C, we first extract one frame per second and then uniformly sample a maximum of 100 frames. For Video Swin, we use the official implementation [48] pretrained on Kinetics-600 [14]. For R(2+1)D, we use the PyTorch [55] implementation pretrained on Kinetics-400 [40], and for C4C we use the CLIP ViT-B/32 encoder trained on MSR-VTT [74].

Although frame and motion match cutting are largely visual tasks, we wondered whether audio carries any signal that can be used for matching. Therefore, we used YAMNet (YN) [77], pretrained on the AudioSet dataset [26] by feeding 16 kHz mono audio from the video and taking the average over the 1024-dimensional embeddings produced for each 0.48 seconds.

Finally, we also consider the concatenation of each image and video encoder with YN. These audio-visual encoders YN-X are constructed by concatenating the YN embeddings with those for X, where X is either an image encoder or a video encoder (e.g. YN-CLIP).

5.2.2 Experiment 1 Results

We present the results in Table 1. We use a simple random baseline method that assigns a random score between 0 and 1 for each candidate pair. The expected AP of this random predictor represents the positive prevalence rate (see more details in Supplementary 3). We include the five heuristics in the table to serve as an additional baseline.

For character frame matching, all visual encoders outperform the random predictor and heuristics h_1 and h_2, while h_3 outperforms CLIP and R(2+1)D. The compound scaling methodology in EN7 [65] tends to produce representations that capture additional object details, which may explain why it performs best for this task (see Supplementary 1 for a description of the nuanced criteria for this task).

Audio (YN) doesn't perform well in isolation, but audio-visual encoders outperform their visual-only counterparts in most cases. Anecdotally, we have observed cases where matching pairs have similar background music, but this pattern is not very consistent.

Motion match cutting is a more challenging and subjective task relative to character frame (see Sec. 4.3.2). All methods beat the random predictor, video encoders perform

Table 2. Experiment 2: Metric Learning Results.

	Character frame		Motion	
Encoder	AP_{val}	AP_{test}	AP_{val}	AP_{test}
CLIP	0.231±0.009	0.321	0.112±0.001	0.138
RN50	0.258±0.010	0.343	0.121±0.005	0.133
EN7	0.283±0.010	**0.373**	0.139±0.005	0.132
R(2+1)D	0.219±0.006	0.261	0.173±0.001	**0.217**
Swin	0.255±0.006	0.363	0.143±0.002	0.170
C4C	0.273±0.010	0.360	0.119±0.001	0.144
YN	0.115±0.010	0.102	0.122±0.001	0.124

better than image encoders, and R(2+1)D achieves the best results amongst them. This makes sense since video encoders are capable of producing representations that capture motion. Also, both h_4 and h_5 do well, as the underlying models, Farnebäck and RAFT, were both specifically designed to capture motion via optical flow. Audio doesn't appear to carry any additional signal for this task.

For both match types, we are able to train models with improved retrieval quality over the heuristic-based approaches. The downside is that we cannot easily use a classifier with ANN methods, as discussed in Sec. 3.4. In the next experiment we explore the possibility of retaining the same level of retrieval quality while using a method that is amenable to ANN methods.

5.3. Experiment 2: Metric Learning

Experiment 1 demonstrated that binary classification is an improvement over heuristics, but is not scalable for inter-movie matching. A suitable approach to achieve this is metric learning, which maps a feature vector to a new embedding vector that can be directly searched through using nearest neighbor methods (rather than running a classifier for each pair). Here we take fixed shot encodings from Experiment 1, and learn a new embedding space via noise contrastive estimation [28]. We leverage the `pytorch-metric-learning` library [52], with NTXentLoss (a.k.a. InfoNCE) [61], [69] and TripletMarginMiner with hard triplets and cosine similarity. The transformed embeddings can be indexed and retrieved with ANN methods as discussed in Sec. 3.4.

We experiment with embeddings from three image encoders: CLIP [57], RN50 [30], and EN7 [65], three video encoders: R(2+1)D [67], Swin [49], and C4C [50], and one audio encoder: YN [77]. For both character frame and motion matching, we use a 2-layer MLP with leaky ReLU [73] activation and train with Adam [41] optimizer. A set of hyperparameters are separately tuned for frame matching and motion matching datasets and are used across experiments of all encoders. More details for this experiment can be found in Supplementary 4.

The results are shown in Table 2. For both frame and motion matching, we find that metric learning is able to achieve higher retrieval quality relative to binary classification, while providing superior inference scalability. EN7 and R(2+1)D perform the best for character frame and motion matching (consistent with the previous experiment).

6. Limitations and Discussion

One limitation of our work is that we have only addressed two specific types of match cutting. We hope to extend this work to other types of match cuts, such as action, sound, light, subject matter, or a broad generalization.

Although our heuristics were designed to aid annotation efforts and are not our main contribution, we will discuss some limitations with them. For frame matching, heuristics 1 was designed to be a crude heuristic and suffers from a lack of spatial alignment between faces. Heuristics 2 and 3 are fairly robust to character framing, though we observe that they can struggle with closeups of faces, where matching face key points may be more suitable. Heuristic 4 and 5 for motion matching tend to surface shots with similar camera movement (as opposed to action). This occurs because the background covers more pixel area than the foreground in most shots. We hope to address this in the future with foreground-background segmentation. Also, our choice of averaging of optical flow outputs over multiple frames discards temporal information.

Furthermore, our proposed frame matching system is limited to only human characters. Non-human objects and shapes can also make for very interesting matches. Lastly, we acknowledge that there is a domain gap between the datasets the models in our experiments were pretrained on and Hollywood produced movies.

7. Conclusions

We have presented a system to take the task of generating good match cut candidates from a manual process akin to finding a needle in a haystack to a semi-automated process that finds "cuts that work" in a fraction of the time.

We summarize our contributions as follows: (1) we present a new and previously unaddressed problem of finding specific match cut types, along with a novel system to identify good candidates. (2) We release a dataset of ∼20k annotated shot pairs for two types: character frame and motion matching. (3) We conduct experiments to evaluate our system using the collected dataset. (4) We release code and embeddings for reproducing our experiments.

While we have presented the first system for finding match cuts, there is plenty of room for improvement. Match cutting is a very challenging task relative to coarse-grained object and action recognition tasks and requires more sophisticated video understanding methods that can capture the nuances.

References

[1] Star Wars: The Rise of Skywalker — final trailer, Oct 2019. Accessed 8 November 2021.

[2] Abien Fred Agarap. Deep learning using rectified linear units (relu). *arXiv preprint arXiv:1803.08375*, 2018.

[3] Humam Alwassel, Dhruv Mahajan, Bruno Korbar, Lorenzo Torresani, Bernard Ghanem, and Du Tran. Self-supervised learning by cross-modal audio-video clustering. *Advances in Neural Information Processing Systems*, 33:9758–9770, 2020.

[4] Wes Anderson. Moonrise Kingdom. *Indian Paintbrush, American Empirical Pictures*, 2012.

[5] Ido Arev, Hyun Soo Park, Yaser Sheikh, Jessica Hodgins, and Ariel Shamir. Automatic editing of footage from multiple social cameras. *ACM Trans. Graph.*, 33(4), July 2014.

[6] Max Bain, Arsha Nagrani, Andrew Brown, and Andrew Zisserman. Condensed movies: Story based retrieval with contextual embeddings. *ACCV*, 2020.

[7] Benigni, Roberto. Life Is Beautiful. *Melampo Cinematografica, Cecchi Gori Group*, 1997.

[8] Floraine Berthouzoz, Wilmot Li, and Maneesh Agrawala. Tools for placing cuts and transitions in interview video. *ACM Trans. Graph.*, 31(4), Jul 2012.

[9] Alex Bewley, Zongyuan Ge, Lionel Ott, Fabio Ramos, and Ben Upcroft. Simple online and realtime tracking. In *2016 IEEE International Conference on Image Processing (ICIP)*, pages 3464–3468, 2016.

[10] Gayatri Bhat and Avneesh Saluja Melody Dye Jan Florjanczyk. Hierarchical encoders for modeling and interpreting screenplays. *NAACL HLT 2021*, page 1, 2021.

[11] G. Bradski. The OpenCV Library. *Dr. Dobb's Journal of Software Tools*, 2000.

[12] Leo Braudy. Film: An international history of the medium. *Film Quarterly (ARCHIVE)*, 48(3):59, 1995.

[13] Qiong Cao, Li Shen, Weidi Xie, Omkar M Parkhi, and Andrew Zisserman. Vggface2: A dataset for recognising faces across pose and age. In *2018 13th IEEE international conference on automatic face & gesture recognition (FG 2018)*, pages 67–74. IEEE, 2018.

[14] Joao Carreira, Eric Noland, Andras Banki-Horvath, Chloe Hillier, and Andrew Zisserman. A short note about kinetics-600. *CoRR*, abs/1808.01340, 2018.

[15] Brandon Castellano. Breakthrough/pyscenedetect: Python and opencv-based scene cut/transition detection program amp; library.

[16] Qifeng Chen and Vladlen Koltun. Full flow: Optical flow estimation by global optimization over regular grids. *CVPR*, 2016.

[17] Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, pages 785–794, 2016.

[18] Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. A simple framework for contrastive learning of visual representations. In *International conference on machine learning*, pages 1597–1607. PMLR, 2020.

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

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

[21] Pete Docter. Up. *Walt Disney studios motion pictures*, 2013.

[22] J.S. Douglass and G.P. Harnden. *The Art of Technique: An Aesthetic Approach to Film and Video Production*. Allyn & Bacon, 1996.

[23] Gunnar Farnebäck. *Two-Frame Motion Estimation Based on Polynomial Expansion*, volume 2749. Springer, Berlin, Heidelberg, 2003.

[24] Christoph Feichtenhofer, Haoqi Fan, Jitendra Malik, and Kaiming He. Slowfast networks for video recognition. *ICCV*, 2019.

[25] Ohad Fried, Ayush Tewari, Michael Zollhöfer, Adam Finkelstein, Eli Shechtman, Dan B Goldman, Kyle Genova, Zeyu Jin, Christian Theobalt, and Maneesh Agrawala. Text-based editing of talking-head video. *ACM Transactions on Graphics*, 2019.

[26] Jort F Gemmeke, Daniel PW Ellis, Dylan Freedman, Aren Jansen, Wade Lawrence, R Channing Moore, Manoj Plakal, and Marvin Ritter. Audio set: An ontology and human-labeled dataset for audio events. In *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 776–780. IEEE, 2017.

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

[28] Michael Gutmann and Aapo Hyvärinen. Noise-contrastive estimation: A new estimation principle for unnormalized statistical models. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, volume 9, pages 297–304, Chia Laguna Resort, Sardinia, Italy, May 2010. PMLR.

[29] Andrey Guzhov, Federico Raue, Jörn Hees, and Andreas Dengel. Audioclip: Extending clip to image, text and audio. *arXiv preprint arXiv:2106.13043*, 2021.

[30] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CVPR*, 2016.

[31] Rachel Heck, Michael Wallick, and Michael Gleicher. Virtual videography. *ACM Transactions on Multimedia Computing Communications and Applications (TOMCCAP)*, 3(1):4, 2007.

[32] Berthold K.P. Horn and Brian G. Schunck. Determining Optical Flow. In James J. Pearson, editor, *Techniques and Applications of Image Understanding*, volume 0281, pages 319 – 331. International Society for Optics and Photonics, SPIE, 1981.

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

[34] Qingqiu Huang, Yu Xiong, Anyi Rao, Jiaze Wang, and Dahua Lin. MovieNet: A holistic dataset for movie understanding. In *Proceedings of the European Conference on Computer Vision (ECCV)*, 2020.

[35] Qingqiu Huang, Yuanjun Xiong, Yu Xiong, Yuqi Zhang, and Dahua Lin. From trailers to storylines: An efficient way to learn from movies, 2018.

[36] Tanuj Jain, Christopher Lennan, Zubin John, and Dat Tran. Imagededup. `https://github.com/idealo/imagededup`, 2019.

[37] Longlong Jing and Yingli Tian. Self-supervised visual feature learning with deep neural networks: A survey. *IEEE transactions on pattern analysis and machine intelligence*, 43(11):4037–4058, 2020.

[38] Jeff Johnson, Matthijs Douze, and Hervé Jégou. Billion-scale similarity search with gpus. *IEEE Transactions on Big Data*, 2019.

[39] Mahdi M Kalayeh, Nagendra Kamath, Lingyi Liu, and Ashok Chandrashekar. Watching too much television is good: Self-supervised audio-visual representation learning from movies and tv shows. *arXiv preprint arXiv:2106.08513*, 2021.

[40] Will Kay, João Carreira, Karen Simonyan, Brian Zhang, Chloe Hillier, Sudheendra Vijayanarasimhan, Fabio Viola, Tim Green, Trevor Back, Paul Natsev, Mustafa Suleyman, and Andrew Zisserman. The kinetics human action video dataset. *CoRR*, abs/1705.06950, 2017.

[41] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[42] 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.

[43] Stanley Kubrick and Arthur C. Clarke. 2001: A space odyssey. *Metro-Goldwyn-Mayer (MGM), Stanley Kubrick Productions*, 1968.

[44] Pinelopi Papalampidi Frank Keller Mirella Lapata. Movie summarization via sparse graph construction. 2021.

[45] Mackenzie Leake, Abe Davis, Anh Truong, and Maneesh Agrawala. Computational video editing for dialogue-driven scenes. *ACM Trans. Graph.*, 36(4), July 2017.

[46] Tsung-Yi Lin, Michael Maire, Serge J. Belongie, Lubomir D. Bourdev, Ross B. Girshick, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C. Lawrence Zitnick. Microsoft COCO: common objects in context. *CoRR*, abs/1405.0312, 2014.

[47] Ze Liu, Han Hu, Yutong Lin, Zhuliang Yao, Zhenda Xie, Yixuan Wei, Jia Ning, Yue Cao, Zheng Zhang, Li Dong, Furu Wei, and Baining Guo. Swin transformer v2: Scaling up capacity and resolution. *arXiv:2111.09883*, 2021.

[48] Ze Liu, Jia Ning, Yue Cao, Yixuan Wei, Zheng Zhang, Stephen Lin, and Han Hu. The official implementation of video swin transformer.

[49] Ze Liu, Jia Ning, Yue Cao, Yixuan Wei, Zheng Zhang, Stephen Lin, and Han Hu. Video swin transformer. *arXiv preprint arXiv:2106.13230*, 2021.

[50] Huaishao Luo, Lei Ji, Ming Zhong, Yang Chen, Wen Lei, Nan Duan, and Tianrui Li. Clip4clip: An empirical study of CLIP for end to end video clip retrieval. *CoRR*, abs/2104.08860, 2021.

[51] Nikolaus Mayer, Eddy Ilg, Philip Häusser, Philipp Fischer, Daniel Cremers, Alexey Dosovitskiy, and Thomas Brox. A large dataset to train convolutional networks for disparity, optical flow, and scene flow estimation. *CoRR*, abs/1512.02134, 2015.

[52] Kevin Musgrave, Serge Belongie, and Ser-Nam Lim. Pytorch metric learning. *arXiv:2008.09164*, 2020.

[53] Alejandro Pardo, Fabian Caba, Juan Leon Alcazar, Ali K. Thabet, and Bernard Ghanem. Learning to cut by watching movies. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 6858–6868, October 2021.

[54] Alejandro Pardo, Fabian Caba Heilbron, Juan León Alcázar, Ali Thabet, and Bernard Ghanem. Moviecuts: A new dataset and benchmark for cut type recognition. *arXiv preprint arXiv:2109.05569*, 2021.

[55] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32:8026–8037, 2019.

[56] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. Scikit-learn: Machine learning in python. *the Journal of machine Learning research*, 12:2825–2830, 2011.

[57] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, Gretchen Krueger, and Ilya Sutskever. Learning transferable visual models from natural language supervision. *CoRR*, abs/2103.00020, 2021.

[58] Abhishek Ranjan, Jeremy Birnholtz, and Ravin Balakrishnan. Improving meeting capture by applying television production principles with audio and motion detection. In *26th Annual CHI Conference on Human Factors in Computing Systems, Conference Proceedings, CHI 2008*, pages 227–236, 2008.

[59] Anyi Rao, Jiaze Wang, Linning Xu, Xuekun Jiang, Qingqiu Huang, Bolei Zhou, and Dahua Lin. A unified framework for shot type classification based on subject centric lens. *ECCV*, 2020.

[60] Adria Recasens, Pauline Luc, Jean-Baptiste Alayrac, Luyu Wang, Florian Strub, Corentin Tallec, Mateusz Malinowski, Viorica Pătrăucean, Florent Altché, Michal Valko, et al. Broaden your views for self-supervised video learning. In

Proceedings of the IEEE/CVF International Conference on Computer Vision, pages 1255–1265, 2021.

[61] Kihyuk Sohn. Improved deep metric learning with multi-class n-pair loss objective. In D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 29. Curran Associates, Inc., 2016.

[62] Tomáš Souček and Jakub Lokoč. Transnet v2: An effective deep network architecture for fast shot transition detection. *arXiv preprint arXiv:2008.04838*, 2020.

[63] Christian Szegedy, Sergey Ioffe, Vincent Vanhoucke, and Alexander A Alemi. Inception-v4, inception-resnet and the impact of residual connections on learning. In *Thirty-first AAAI conference on artificial intelligence*, 2017.

[64] Yoshinao Takemae, Kazuhiro Otsuka, and Naoki Mukawa. Video cut editing rule based on participants' gaze in multiparty conversation. In *Proceedings of the Eleventh ACM International Conference on Multimedia*, page 303–306, New York, NY, USA, 2003. Association for Computing Machinery.

[65] Mingxing Tan and Quoc Le. Efficientnet: Rethinking model scaling for convolutional neural networks. In *International Conference on Machine Learning*, pages 6105–6114. PMLR, 2019.

[66] Zachary Teed and Jia Deng. RAFT: recurrent all-pairs field transforms for optical flow. *ECCV*, 2020.

[67] Du Tran, Heng Wang, Lorenzo Torresani, Jamie Ray, Yann LeCun, and Manohar Paluri. A closer look at spatiotemporal convolutions for action recognition. *CVPR*, 2018.

[68] Anh Truong, Floraine Berthouzoz, Wilmot Li, and Maneesh Agrawala. Quickcut: An interactive tool for editing narrated video. In *Proceedings of the 29th Annual Symposium on User Interface Software and Technology*, UIST '16, page 497–507, New York, NY, USA, 2016. Association for Computing Machinery.

[69] Aaron van den Oord, Yazhe Li, and Oriol Vinyals. Representation learning with contrastive predictive coding. *arXiv:1807.03748*, 2019.

[70] Paul Vicol, Makarand Tapaswi, Lluis Castrejon, and Sanja Fidler. Moviegraphs: Towards understanding human-centric situations from videos. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 8581–8590, 2018.

[71] Lana Wachowski and Lilly Wachowski. The Matrix. *Warner Bros*, 1999.

[72] Miao Wang, Guo-Wei Yang, Shi-Min Hu, Shing-Tung Yau, and Ariel Shamir. Write-a-Video: Computational video montage from themed text. *ACM Trans. Graph.*, 38(6), Nov. 2019.

[73] Bing Xu, Naiyan Wang, Tianqi Chen, and Mu Li. Empirical evaluation of rectified activations in convolutional network. *arXiv:1505.00853*, 2015.

[74] Jun Xu, Tao Mei, Ting Yao, and Yong Rui. Msr-vtt: A large video description dataset for bridging video and language. IEEE International Conference on Computer Vision and Pattern Recognition (CVPR), June 2016.

[75] Pavel Yakubovskiy. Segmentation models pytorch. `https://github.com/qubvel/segmentation_models.pytorch`, 2020.

[76] Shen Yan, Xuehan Xiong, Anurag Arnab, Zhichao Lu, Mi Zhang, Chen Sun, and Cordelia Schmid. Multiview transformers for video recognition. *arXiv:2201.04288*, 2022.

[77] Hongkun Yu, Chen Chen, Xianzhi Du, Yeqing Li, Abdullah Rashwan, Le Hou, Pengchong Jin, Fan Yang, Frederick Liu, Jaeyoun Kim, and Jing Li. TensorFlow Model Garden. `https://github.com/tensorflow/models`, 2020.

[78] C. Zach, T. Pock, and H. Bischof. A duality based approach for realtime tv-l1 optical flow. In Fred A. Hamprecht, Christoph Schnörr, and Bernd Jähne, editors, *Pattern Recognition*, pages 214–223, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg.

[79] Robert Zemeckis. Forrest gump. *Paramount Pictures*, 1994.