

IntentVizor: Towards Generic Query Guided Interactive Video Summarization

Supplementary Material

Guande Wu^{*} Jianzhe Lin^{*} Claudio T. Silva
New York University
{guandewu, jianzhelin, csilva}@nyu.edu

Appendix A: Visual Query Dataset

Algorithm 1 Visual Query Generation

Require: $A = \{\alpha_1, \alpha_2, \alpha_T\}$ as the sequence of the shot’s semantic tags
Require: $S = \{s_1, s_2, s_P\}$ as the summary shots of the video
Require: k as the target visual query number
Ensure: Q as set of representative query shots in S .
 $W \leftarrow \text{Matrix}(P, P)$
 $i = 0$
while $i < P$ **do**
 $j = 0$
 while $j < P$ **do**
 $W[i, j] = \text{SemanticIOU}(s_i, s_j)$
 $j \leftarrow j + 1$
 end while
 $i \leftarrow i + 1$
end while
 $G \leftarrow \text{ConstructGraphFromWeights}(W)$
 $D \leftarrow \text{EigenVectorCentrality}(G)$
 $S' \leftarrow \text{RankByCentrality}(S, D)$
 $Q \leftarrow S'[:k]$

In this appendix, we present the details of our proposed visual query dataset. We build our visual-query dataset on the text-query dataset. For each annotated summarization, we employ the eigenvector centrality as the criteria to pick the most representative shots as the query shots.

Query Generation

We employ eigenvector-centrality method to select the most representative shots as the query shots. We detail our query generation in Algorithm 1. For each video summary set S , we first compute the pairwise semantic IOU [6] of the shots in the summary set S . In order to do it, we first create a zero-valued matrix W of $P \times P$. Then we loop over

the summary set to compute the semantic IOU between every pair of the shots. The function *SemanticIOU* follows the implementation of [6]. The result will be stored in the matrix W . Then, we build a weighted undirected graph G based on the obtained IOU weight matrix W . We implement the function *ConstructWeightedGraph* to fulfill it. Then, we can compute the eigen-vector centrality of each vertex by a Python package called Networkx. We wrap their implementation in the function *EigenVectorCentrality*, which outputs a dictionary D for the eigen-vector centrality values of the all shots. Finally, we can rank the vertices by the eigen-vector centrality and pick the top- k as the query shots. We implement a function *RankByCentrality* to fulfill it. The result of the function is a ranked list S' of the shots in S . In practice, we set $k = 5$.

Dataset Examples

We present some samples in the dataset as follows. Specifically, we put the visual query on top of the full video summary as shown in Figure 1-4. The four examples are collected from the four different videos.

Figure 1 shows the visual query and summary for Video-1 when the query words are “Food” and “Men”. We can find the five visual queries include the scenarios when a man is eating the food, which follows the textual query and are close to the full summary.

Similarly, we can find that in Figure 2, the visual summary also follows both the textual query and the full video summary. The query words are “Garden” and “Party”. We can find that the query shot #1, #3, #4, #5 are all related to the “Garden” while the query shot #3 and #5 are related to “Party”. We notice that the query #2 is a car driving scenario, which is not related to either “Party” or “Garden”. However, by checking the full video summary, we find that there are a lot of car driving scenarios in the full video summary. Thus, the query shot #2 is closely related to the full video summary.

In Figure 3, the query words are “Hands” and “Room”. We find that the query #2, #3 and #4 have a hand in the picture, while all of the query shots are captured in a room.

^{*}equal contribution

In Figure 4, the query words are “Sun” and “Tree”, we find that the queries are taken on the sunny day, which relates to the query word “Sun”. Also, the query shot #1, #2, #4 and #5 have the trees in the foreground.

Appendix B: Prototype Implementation

In order to validate the interactivity of our proposed method, we implement a visual interface allowing the user to interact with the model. We present the implementation details below. Our prototype consists of a client-side interface and server-side backend. The interface is built purely on browser with D3.js [1], React.js and Typescript. Specifically, we rely our whole interface on React.js and implement the visualization by D3.js. We implement the Restful APIs with Flask on the server-side.

Client-Side Interface

Our client interface is built with D3.js and React.js purely on Typescript. The interface consists five major components, i.e, Initial View, Summary View, Intent View, Preview View, Query View and Evaluation View. Below, we will first enumerate the views before presenting a user scenario.

Initial View

Initial View allows the user to select the video, textual queries and the model checkpoint. After the user submit the query, the system will automatically direct to the output of the model.

Summary View

Summary View presents the predicted summary of the corresponding video. The predicted selection probability (shot score) is visualized with a bar chart. We highlight the summary shot by the blue color. Since the videos are extremely long, as they span 3-5 hours. The user is unable to get any detail when viewing the summary overview. We provide a focus view on top of the summary overview. The focus view is also a bar chart while the time range is limited. The user can brush on the overview chart to control the time range in the focus view. This “focus+context” interaction style can enable the user gain the detail and overall information simultaneously.

Intent View

Intent View lists all the intents with their probability (weight), selection probability (intent-based shot score) and representative samples. Since the representative samples are collected based on the shot probability, they can provide a visual feedback to the user and offer a impression

of the specific intent. We allow the user interact with the intent probability and express their subtle intention. For example, in the use case described in our paper, the user inputs the “Food” and “Room”. The user can find that intent #18 corresponds to the food cooking when intent # 11 focuses more on the food storage. Though the user may want to focus more on the food storage rather than food cooking. Though this type of textual query is not included in the original dataset, the user express his intention by controlling the intent probability. The case proves that our method can enable the finer-grained adjustment compared with the previous methods.

Preview View

The Preview View shows the GIF-formatted shot. The user can hover on the shot in each intent to get its preview.

Query View

Query View is exactly same with the initial view. However, it allows the user to change the query parameters, i.e, query concepts, video and model checkpoint to re-run the model.

Evaluation View

Evaluation view presents the quantitative evaluation of the running result. Currently we compare the user-generated summary with the ground-truth to get F-1 measure, precision and recall. We encode the result value by both the arc degree and the color. The green means the result is higher than 40%, the yellow means the result is higher than 20%, the red means the result is lower than 20%. The user can adjust the intent probability, which will generate a new summary. The new summary will be automatically sent to the back-end to get the quantitative evaluation, which is then shown in this view.

Server-Side Backend

We implement our server-side APIs with Flask. We provide three groups of APIs to support the user interaction.

Model Preparation API

The model preparation API provides the front-end interface with necessary model information before running the model. Specifically, the API returns the available model checkpoints, video keys. We are planned to support the user-uploaded video in the future.

Model Inference API

The model inference API takes the two query words, one video id string and one model checkpoint id. The API will



Visual Query



Full Video Summary

Figure 1. “Food” and “Men” on Video 1



Visual Query



Full Video Summary

Figure 2. “Garden” and “Party” on Video 2



Visual Query



Full Video Summary

Figure 3. “Hands” and “Room” on Video 3



Visual Query



Full Video Summary

Figure 4. “Sun” and “Tree” on Video 4

returns the results of the intent-network (intent probability) and scoring-network (shot probability). The calculation of the overall shot probability and the selection of the summary shots are conducted on the front-end.

Video Shot Frame API

Video shot frame API takes a video key and a shot index to return the corresponding frame image of the shot. The shot can have multiple frames, we employ a random strategy to select a frame. The API is designed for selecting the



Figure 5. Prototype Overview. **A: Summary View** presents two temporal bar charts, which shows the overall scores and the summarized shots. The bottom bar chart shows the overview of all the shots while the top bar chart zooms into the detail decided by the brush in the bottom chart. **B: Intent View** list all the intents with their probability, shot scores and representative samples. The samples are selected with the highest score. **C: Preview View** plays a GIF of the user-hovering shot. In this case, the user hovers on the highlighted shot in intent #12, which includes a room scenario. **D: Query View** allows the user to change the query and makes the model run again. **E Evaluation View** shows the quantitative result of the summary.

API	Method	Parameters	Parameter Types	Response	Response Type
Model Preparation	GET	N/A	N/A	list of checkpoints	JSON
Model Inference	POST	word1 word2 video checkpoint	string string string string	intent probability and shot selection probability	JSON
Video Shot Frame	GET	video shot	string integer	Frame Image	Image/PNG
Video Shot GIF	GET	video shot	string integer	Shot GIF	Image/GIF
Quantitative Evaluation	POST	video summary	string Array<integer>	F1, Precision and Recall	JSON

Table 1. Server-Side Backend APIs

representative samples of the intents.

Video Shot GIF API

Video shot GIF API returns a GIF-formatted shot according to the input video key and shot index. The GIF is clipped from the original video via OpenCV. The API is designed for Preview View in the interface.

Quantitative Evaluation API

Quantitative evaluation API takes a summary to generate the quantitative evaluation result. The evaluation includes

F-measure, precision and recall. The API is designed for the user-generated summary.

API Table

We list our APIs in Table 1 with their HTTP methods, parameters and responses.

User Scenario

Below, we present a user scenario to demonstrate the interaction between the user and interface as well as the connection between the interface and the back-end server. Before rendering the webpage, the interface will first send a

request to Model Preparation API to get the necessary information for the initial view. Then the initial view will be presented to the user. The user can first input the query words, video, and model checkpoint in the initial view. Here the user inputs the "Food" and "Room" as the words. Then the user chooses the Video-3 and model "default". After submitting the query information, the interface will send a GET request to Model Inference API and render the whole page based on the response. We employ an asynchronous rendering mechanism to reduce the latency. The asynchronous mechanism allows the interface to first renders the Summary View and Intent View before rendering the representative frames and Quantitative Evaluation View. During the process, the interface selects the representative samples based on the shot selection probability (shot score) and requests the Video Shot Frame API to get the corresponding image. It also sends the generated summary to Quantitative Evaluation API to render the Evaluation View. After rendering all the views, the user can brush on the overview bar chart and the brush operation will determinate the temporal range in the focus bar chart. The user can view different intents and understand them by the representative samples. Then the user can adjust the intent probability based on his preference. The adjusted probability distribution will generate a new summary, which will change the Summary View and Quantitative View. To get the new evaluation result, the interface sends a request to Quantitative Evaluation API again. To view the specific shot, the user can hover on the shot. The operation will make the interface send a request to Video Shot GIF API, whose response will be rendered in Preview View.

We also implement a prototype for the visual-query task as shown in Fig. 6. The user can select a series of video shots based on the start and end times. Such a method allows the user to flexibly choose the queries and enhances the generalization ability of the method. After the user finish the query setting, the interface can send the user queries to the backend server, which predicts the user intents and the initial video summary. Then, the front-end interface renders the response and supports the user interaction, similar to the textual-query prototype.

Appendix C: Implementation Details

Below, we will present the implementation details of our IntentVizor framework. As described in the main paper, our method consists two modules, i.e., intent module and summary module, both relying on Granularity-Scalable Pathway (GS-Pathway) and Ego Graph Convolutional Network (Ego-GCN). Thus, we will first describe the implementations of our proposed GS-Pathway and Ego-GCN. Then, we will further detail the intent and summary modules.

Granularity-Scalable Pathway (GS-Pathway)

Our GS-Pathway consists two pathways of different temporal granularity. Each pathway consists two convolutional layers followed by the max pooling layers respectively. We list the hyper-parameters in Table 2. The processed features will be fed into the E-GCN to predict the summary and user intent. According to Table 2, the coarse pathway has a larger stride and outputs a shorter feature sequence with a length of $\lceil \frac{T}{16} \rceil$, where T is the original video length. By contrast, the fine pathway has a smaller stride and outputs a longer feature sequence with a length of $\lceil \frac{T}{4} \rceil$. Besides, to extract the shot-level features, We employ the Resnet features provided by Xu et al. [8]

Ego-GCN

The Ego-Graph backbone first maps the intent/query into a space with same dimension as segment feature dimension (256 for fine pathway and 1024 for coarse pathway). We employ three GCN layers in the summary module and two layers in the intent module. Each GCN layer consists two consequential sub-layer, each of which perform one convolution operation on the constructed graph. For each GCN layer, we also add a shortcut between the input and output. Our constructed graph consists of three types of edges. There is no parameter for the temporal edge and intent-segment edge. For the semantic edge, we set the number of edges for one vertex as 8.

Summary Module

Our summary module feeds the intent embedding into the GS-Pathway and Ego-GCN consequently. After that, we the processed video segment features will be fed into our summary head. The intent head consists of three modules, i.e., Local-GCN, dot product relevance module and MLP.

The local GCN recovers the segment-level features to the shot-level. We construct a graph for each segment where the segment vertex is connected with all shot vertices. To build the vertices, we map the segment feature and shot feature to a mutual space of 512D by a linear layer. Following Ego-GCN, we also employ temporal, semantic edges. We set the numbers of semantic edges for one shot vertex as 4 for fast pathway and 10 for slow pathway.

We employ a relevance module to compare the recovered shot-level features and the intent embedding. We first map them to a mutual space of 1024D by two linear layers respectively. Then, we perform dot product on the two generated vectors. The result will be fed into a three-lay MLP with a hidden dimension of 1024. The MLP is followed by a Sigmoid to produce the shot selection probability.

Intent Module

The intent module predicts a weighted combination of the learned basis intent shown in Sec. 3.1.1. We agree that



Figure 6. The prototype snapshot for the visual-query based video summarization. There are two pages for the user. Firstly, **Visual Query Selection** page allows the user selects the query shots based on the input video. The query shots are specified based on the user-inputted start and end times. Then, **Summary Generation** page renders the generated summaries and supports the user interaction in a way similar to the textual-query prototype.

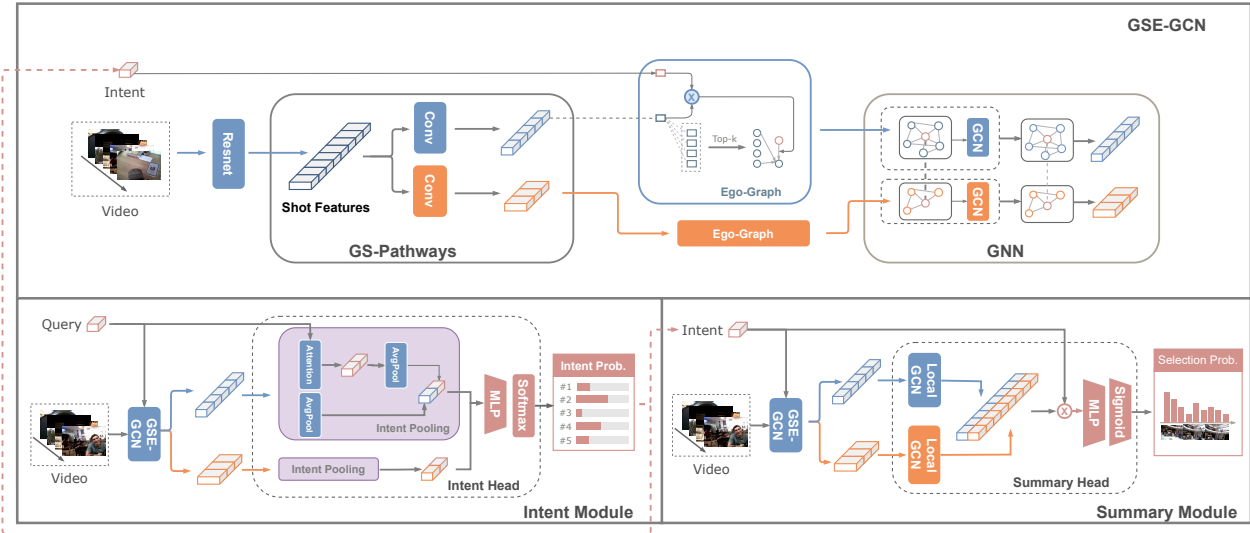


Figure 7. **GSE-GCN** exploits two notions i.e, GS-Pathway and Ego-Graph. The input video will be processed by two convolutional networks to produce two segment-level feature sequence of coarse and fine granularity. Then, each sequence will be processed to generate a Ego-Graph, where the intent/query vertex is an ego-vertex with all the video segments are connected. After feeding the graph into GCN, the two pathways will be produce the corresponding segment-level features. **Intent Head** pools the segment features into a distributed representation, which will be processed by a MLP with softmax to produce the intent probability. **Summary Head** exploits the local-GCN module to produce the shot-level features, which will be used to predict the shot selection probability.

the intents can be non-overlapping with the visual/textual queries when the inputted video is very different from the training data. However, the variety of queries in the dataset ensures the generalizability of the trained models. There are two intent modules with similar structure to support the textual and visual queries respectively.

Intent Module for Textual Query

The intent module for textual query takes input of two query words (concepts) to predict the intent probability of the basis intents. Similar to [8], we employ the publicly available GLOVE word embedding [5] to represent the query words. The inputting query words are firstly concatenated into one embedding vector(query embedding). Then, fol-

Layer	Coarse Pathway				Fine Pathway			
	Kernel	Stride	Channel	Output Size	Kernel	Stride	Channel	Output Size
Conv1	5	8	1024	[L//8, 1024]	5	1	256	[L//2, 256]
MaxPool1	2	1	1024	[L//8, 1024]	2	2	256	[L//2, 256]
Conv2	5	1	1024	[L//8, 1024]	5	1	256	[L//2, 256]
MaxPool2	3	2	1024	[L//16, 1024]	2	2	256	[L//4, 256]

Table 2. The hyperparameter setting of the granularity-scalable pathways. L refers to the length of the original video.

Dataset	Method		Video-1			Video-2			Video-3			Video-4			Average		Std Dev				
	Study	Model	Pre	Rec	F1	Pre	Rec	F1	Pre	Rec	F1	Pre	Rec	F1	Pre	Rec	F1	Pre	Rec	F1	
Textual	Default	Full Model	62.19	45.23	51.27	50.43	57.81	53.48	73.45	53.56	61.58	28.24	56.47	37.25	53.58	53.27	50.90	19.33	5.64	10.12	
		T+T	53.81	38.89	44.19	42.31	49.55	45.82	60.33	43.97	50.57	22.84	45.66	30.11	44.82	44.52	42.67	16.44	4.42	8.80	
	Ego-GCN	T+E	58.24	42.17	47.89	43.4	49.76	46.01	70.06	50.98	58.66	24.28	48.63	32.05	48.995	47.89	46.15	19.76	3.93	10.93	
		E+T	55.98	39.94	45.59	39.85	45.58	42.22	64.15	46.75	53.77	28.39	56.76	37.44	47.09	47.26	44.76	16.04	7.00	6.88	
	Local-GCN	Upsampling	56.72	40.65	46.31	35.61	40.91	37.8	39.95	28.68	33.18	19.89	39.67	26.21	38.04	37.48	35.88	15.14	5.89	8.43	
		Transpose	54.28	38.56	44.05	48.71	55.8	51.64	62.28	45.37	52.18	24.85	49.91	32.84	47.53	47.41	45.18	16.11	7.28	9.02	
	GS-Pathway	Coarse-Only	56.00	40.45	45.96	42.42	48.42	44.88	63.40	46.21	53.14	27.76	55.57	36.63	47.40	47.66	45.15	15.71	6.25	6.76	
		Fine-Only	58.45	42.3	48.04	47.9	54.91	50.8	66.63	48.51	55.82	27.73	55.39	36.57	50.18	50.28	47.81	16.81	6.17	8.15	
			Shot Feature	56.00	40.45	45.96	42.42	48.42	44.88	63.40	46.21	53.14	27.76	55.57	36.63	47.40	47.66	45.15	15.71	6.25	6.76
		Fusion Stage	MiddleFusion	56.82	40.97	46.59	46.27	53.1	49.1	69.98	50.32	57.85	25.05	50.24	33.08	49.53	48.66	46.66	18.98	5.29	10.26
		LateFusion	54.31	39.2	44.56	43.89	50.29	46.53	65.17	47.51	54.63	27.4	54.9	36.17	47.69	47.98	45.47	16.08	6.60	7.58	
	Intent Module	Video Agnostic	57.5	41.26	47	48.83	55.97	51.78	70.92	51.7	59.45	22.98	46.17	30.36	50.06	48.78	47.15	20.21	6.42	12.31	
Visual	Default	Video Attention	57.29	41.02	46.75	44.46	50.98	47.16	71.87	52.41	60.26	23.4	46.99	30.9	49.26	47.85	46.27	20.55	5.10	12.01	
		Full Model	58.17	44.91	49.43	42.52	52.69	46.64	65.45	51.92	57.49	21.15	49.23	29.19	46.82	49.69	45.69	19.61	3.51	11.92	
	Comparative Study	Attention Baseline	45.01	33.96	37.71	38.86	48.01	41.09	57.7	48.75	50.66	18	41.5	24.75	39.89	43.06	38.55	16.57	6.88	10.71	
		Linear Baseline	59.24	45.33	49.75	21.49	26.71	23.62	56.09	44.42	49.22	14.44	33.1	19.77	37.82	37.39	35.59	23.14	9.04	16.12	
		Transferring Study	Transferring	56.15	43.07	47.53	45.23	56.19	49.69	63.8	50.73	56.12	23.4	54.31	32.26	47.15	51.08	46.4	17.57	5.80	10.11

Table 3. Full experiment result. For the models of Ego-GCN (ablation) study, T(E)+T(E) means that the model uses Transformer (Ego-GCN) as the intent module and Transformer (Ego-GCN) as the summary module respectively. For example, T+E means that the model uses Transformer as the intent module and Ego-GCN as the summary module.

lowing the summary module, we employ the GSE-GCN to model the relationships between the query embedding and the segments. However, we use the query embedding as the ego vertex rather than the intent embedding. We use the same setting as we set the number of semantic edge as 8.

Then the GSE-GCN-produced segment-level feature sequence is used by the intent head to generate the intent probability distribution. To fulfill it, the produced feature sequence is pooled temporally to obtaining a vector attending to both the video content and the query. We call the pooling module as intent pooling. The intent pooling consists two modules, i.e, attention pooling and average pooling. The average pooling performs the average pooling on the segment-level feature sequence to get a vector. By comparison, the attention pooling first inputs the queries' word embeddings as the "query" in multi-head attention module. Then, it uses the feature sequence as the "key"- "value" pair. We set the number of heads as 5. The output of the attention module has two elements corresponding to two query words. The two elements are pooled to get an average vector. Then, we concatenate the two vectors of two pathways (overall 4) and feed the concatenated vector into a three-layer MLP with a hidden dimension of 2048. Finally, we use a Softmax module to get the probability of each basis intent.

Intent Module for Visual Query

The input of shot query intent module is the visual query consisting of P shot features $\{s_1, s_2, \dots, s_P\}$. The module follows a similar structure as the text query module while it deviates in the Ego-Graph structure. The visual query comprises denser information with a higher feature dimension (2048D) compared with the text query (300D). Merging the visual queries together may incur a heavy loss of information. Thus, the visual query module takes the query shots as the individual vertices, and all feature vertices are connected with the queries. Other parameters are identical to the intent module for textual query dataset.

Baselines for the Visual Query Task

We propose two baselines for the visual-query task in Sec. 4.2.2. Below, we will describe the implementation details for them.

Attention Baseline

The attention baseline employs the attention mechanism to evaluate which shot is mostly related to the visual queries. The structure is similar to the QAM module (Query-focused Attention Module) in [3]. The calculated attention values are multiplied with the video shot features to produce the query-aware video features. We stack three attention layers with the residual connection to promote the representation

capability. The video features produced by the last layer is fed into a three-layer MLP to compute the selection probability for each video shot.

Transformer Baseline

The transformer baseline employs the popular transformer structure and follows [7]. We use the video shot and query shot as two different tokens. Then, we concatenate them and feed into a transformer encoder, which consists of three layers. Then a three-layer MLP is used to predict selection probability of each shot.

Training Setting

We set the intent number as 20 and the embedding dimension as 128. Models are trained by an Adam optimizer with a base learning rate of $1e-4$ and a weight decay of $6e-5$. We employ a warm-up strategy [2] to linearly increase the learning rate from 0 to the base learning rate in 10 epochs. After that, we reduce the learning rate to one-tenth of the previous value every twenty epochs. We set the shifted ReLU threshold as 0.05. We set the number of epochs as 120. We set the batch size as 2.

Fighting Overfitting

Overfitting is a critical issue in training the deep neural networks, especially when the networks comprises a huge number of parameters. We fight the overfitting issue by both the design of network and dataset. 1. The design of the network relieves the overfitting problem. Our proposed scalable granularity can avoid overfitting by reducing the number of shots and the size of features (Sec. 3.2.1). Also, we propose to construct the graph edges based on the pre-designed heuristics to further eliminate the parameter size (Sec. 3.2.2). The standard training techniques (including early stop, momentum, gradient clip) also helps relieve the overfitting. 2. Though the dataset contains only four videos, each video is paired with a far larger number of different queries [6]. Also, the long videos contain a wide variety of actions/events, diversifying the dataset [4].

Appendix D: Full Experiment Result

In this appendix, we present the full experiment results of the ablation study in Table 3. We group the methods based on the dataset and the belonging study. We also compute the standard deviation for each method.

References

- [1] Michael Bostock, Vadim Ogievetsky, and Jeffrey Heer. D³ data-driven documents. *IEEE Transactions on Visualization and Computer Graphics*, 17(12):2301–2309, 2011. 2
- [2] Priya Goyal, Piotr Dollár, Ross Girshick, Pieter Noordhuis, Lukasz Wesolowski, Aapo Kyrola, Andrew Tulloch, Yangqing Jia, and Kaiming He. Accurate, large mini-batch sgd: Training imagenet in 1 hour. *arXiv preprint arXiv:1706.02677*, 2017. 9
- [3] Pin Jiang and Yahong Han. Hierarchical Variational Network for User-Diversified & Query-Focused Video Summarization. In *Proceedings of the 2019 on International Conference on Multimedia Retrieval*, pages 202–206, Ottawa ON Canada, June 2019. ACM. 8
- [4] Yong Jae Lee, Joydeep Ghosh, and Kristen Grauman. Discovering important people and objects for egocentric video summarization. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1346–1353. IEEE, 2012. 9
- [5] Jeffrey Pennington, Richard Socher, and Christopher D Manning. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543, 2014. 7
- [6] Aidean Sharghi, Jacob S. Laurel, and Boqing Gong. Query-Focused Video Summarization: Dataset, Evaluation, and a Memory Network Based Approach. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2127–2136, Honolulu, HI, July 2017. IEEE. 1, 9
- [7] Chen Sun, Austin Myers, Carl Vondrick, Kevin Murphy, and Cordelia Schmid. Videobert: A joint model for video and language representation learning. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 7464–7473, 2019. 9
- [8] Shuwen Xiao, Zhou Zhao, Zijian Zhang, Xiaohui Yan, and Min Yang. Convolutional Hierarchical Attention Network for Query-Focused Video Summarization. *Proceedings of the AAAI Conference on Artificial Intelligence*, 34(07):12426–12433, Apr. 2020. 6, 7