Compound Memory Networks for Few-shot Video Classification

Linchao Zhu and Yi Yang

CAI, University of Technology Sydney, NSW
Linchao.Zhu@student.uts.edu.au; Yi.Yang@uts.edu.au

Abstract. In this paper, we propose a new memory network structure for few-shot video classification by making the following contributions. First, we propose a compound memory network (CMN) structure under the key-value memory network paradigm, in which each key memory involves multiple constituent keys. These constituent keys work collaboratively for training, which enables the CMN to obtain an optimal video representation in a larger space. Second, we introduce a multi-saliency embedding algorithm which encodes a variable-length video sequence into a fixed-size matrix representation by discovering multiple saliencies of interest. For example, given a video of car auction, some people are interested in the car, while others are interested in the auction activities. Third, we design an abstract memory on top of the constituent keys. The abstract memory and constituent keys form a layered structure, which makes the CMN more efficient and capable of being scaled, while also retaining the representation capability of the multiple keys. We compare CMN with several state-of-the-art baselines on a new few-shot video classification dataset and show the effectiveness of our approach.

Keywords: Few-Shot Video Learning · Video Classification · Memory-augmented Neural Networks · Compound Memory Networks

1 Introduction

Deep learning models have been successfully applied to many tasks, e.g., image classification [16, 25, 29, 8], image detection [22], video classification [12, 24] and machine translation [28, 36]. Convolutional Neural Networks (ConvNets) and Recurrent Neural Networks (RNNs) have become built-in modules in various fields. However, large amounts of labeled training data are required to train a deep neural network. To adapt an existing model to recognize a new category which was unseen during training, it may be necessary to manually collect hundreds of new training samples. Such a procedure is rather tedious and labor intensive, especially when there are many new categories. There is an increasing need to learn a classification model from a few examples in a life-long manner, which is also known as the few-shot learning task [23, 32].

In a few-shot recognition setting, the network needs to effectively learn classifiers for novel concepts from only a few examples. Unlike traditional models

Fig. 1. The setting of the few-shot video classification. There are two non-overlapping datasets in this figure, i.e., meta-training and meta-testing. The meta-training set is for meta-learning and the meta-testing set is for evaluating the generalization performance on novel categories. The network is trained in an episodic way and each episode has a support set and a query example.

trained on many data samples, the model in a few-shot setting is trained to generalize across different episodes. In contrast to training new classifiers by fine-tuning, we propose a learning to learn approach under the meta learning paradigm [23]. We aim to enable a system to learn how to classify video data into a new category by exploiting a meta-training set. As shown in Figure 1, the meta-training set consists of a number of episodes which mimic the few-shot learning task. In this example, there is only one positive exemplar per class in each episode, indicated by a red rectangle. There is no overlapping category between the training phase and testing phase. During the training phase, the system learns an optimal mechanism that best recognizes queries in all training episodes. When testing, the system directly adopts the learned optimal mechanism to classify each query in testing episodes.

In this paper, we focus on few-shot video representation learning. Videos have more complex structures than images, involving temporal information and more noise, e.g., camera motion, object scales, viewpoints. It is a more challenging task than few-shot image classification. Many videos usually contain hundreds of frames containing various scene dynamics. It could be difficult to understand the concept in a video when only few examples are provided.

We thus propose a compound memory network (CMN) structure for few-shot video classification. Our CMN structure is designed on top of the key-value memory networks [35] for the following two reasons. First, new information can be readily written to memory, which provides our model with better 'memorization' capability. In other words, MANNs are able to store and memorize an example long-term, even though the example has been seen only once. Second,

information stored in the memory module can be memorized for a longer period and can be easily accessed. During training, information in each training episode is gradually accumulated into CMN, which is then used as the learned few-shot video classification mechanism for testing. It is worthwhile highlighting the following aspects of our CMN model.

First, we propose a new notion of compound memory networks with a much stronger representation capability by extending the key memory elements from a 1D vector to a 2D matrix. Standard key-value memory networks use a single vector as the key in each memory slot [19]. Videos are more complex in structure than images and have richer semantic information. Accordingly, we propose to use multiple vectors to enhance the video representation, with each vector being a constituent key. The constituent key are stacked to a matrix to generate the video representation in CMN. These stacked constituent keys work collaboratively in the training phase, providing a larger search space from which to obtain an optimal video representation.

Second, we introduce a series of hidden saliency descriptors as constituent keys in the memory slots of CMN. In many cases, user may be interested in different salient parts of a video. For example, given a video of a birthday party, some users may be more interested in the dancing scene, while others focus on the food and drinks. We propose a multi-saliency embedding algorithm which automatically detects multiple saliencies of interest in any given video. We extend the self-attention mechanism [18, 31] by integrating a newly designed learnable variable to adaptively detect hidden salient genres within a video. The multi-saliency embedding algorithm learns a hidden saliency descriptor for each genre, which is then stacked as a video representation in CMN.

Third, we design a layered memory structure, which vastly improves efficiency while retaining the strong representation capability of CMN. The first layer stores the stacked constituent keys. We design an abstract memory on top of the first layer, which is equipped with reading and writing operations for retrieving and updating the constituent keys. The abstract memory compresses the stacked constituent keys into a vector and vastly improves training and testing efficiency. At the same time, the communication between the two layers ensures that abstract memory is able to retain the information from all constituent keys.

2 Related Work

Few-shot Classification. Early works from Miller et al. [20], Fei-Fei et al. [4] and Lake et al. [17] utilized generative models for one-shot learning. Koch [15] attempted to train a Siamese network in a supervised way. Santoro et al. [23] was the first work to successfully bridge memory-augmented neural works and one-shot learning. They took training examples in an episode as sequential inputs and trained the network to predict the label given previous examples. Vinyals et al. [32] used metric learning for few-shot recognition and utilized the attention kernel to measure the distance. Given a query, the network is trained to "point" to the nearest instance in the support set and the corresponding label is re-

trieved as the prediction. Ravi and Larochelle [21] trained a meta-learner based on Long Short-Term Memory (LSTM) [10] to generate updates for the classifier rather than using gradients. The meta-learner also learns a task-common weight initialization which captures shared knowledge across tasks. Finn et al. [5] used stochastic gradient descent as a meta-learner to update the parameters of the learner, which only learns the weight initialization. Snell et al. [26] applied a similar model to Vinyals [32], but they used Euclidean distance with their embedding function. Hariharan and Girshick [7] proposed the generation of images at testing time to improve few-shot recognition performance. Xu et al. [37] presented a key-value memory network to facilitate few-shot learning by extracting knowledge from external knowledge bases, e.g., noisy web images. However, their setting is not the meta-learning paradigm. These works focus on image few-shot recognition, whereas we aim to learn a few-shot video model, which requires modeling complex video data.

Video Classification. Video classification methods have evolved from using hand-crafted features, e.g., improved dense trajectories [33], to deep models, e.g., two-stream Convolutional Neural Networks (ConvNets) [24, 34], 3D ConvNets [30], two-stream 3D ConvNets [3]. Recurrent Neural Networks have also been utilized to model video sequences [38, 39]. Many efforts have been made to train a video classification model using large amounts of video data, however, it would be expensive to collect large datasets and retrain the classifier for all novel categories. The few-shot video classification task is more realistic in a real-world scenario, where the model will encounter novel categories that are never seen during training. The networks should be trained to adapt to new tasks.

Memory-Augmented Neural Networks. Memory-Augmented neural networks have gained increasing interest with the success of attention mechanism [2], Neural Turing Machine [6], and Memory Networks [35]. In RNNs, the states transferred between the steps can be interpreted as internal memory representations for the inputs. The state vector of the last step is usually used as the final representation for the whole input sequence. The fixed-size vector representation cannot encode long sequences in an effective way. Instead, the attention mechanism retains a sequence vectors as contexts for content-based addressing. The states in RNNs can change quickly over a few steps, while an external memory can retain information over the long term. Neural Turning Machine [6] is a computer-like network augmented with an external memory that can be addressed via content and location. The reading and writing operations are fully differentiable and weight updates through backpropagation are applied to every memory slot. Memory networks [35] and the improved end-to-end memory networks [27] have a large memory component for fact search and retrieval through content-based addressing. Key-value memory networks [19] decompose the memory into key and value parts, introducing a structural memory component to store question-answer pairs in a flexible way. Soft addressing is used in all these works, which is computationally expensive with growth of the memory size. Kaiser et al. [11] recently proposed a key-value memory module which performs hard updating to the memory, and a ranking loss is used to train the model to

make accurate predictions. However, the memory stores only a fixed-size vector for an input, which is not suitable when the input is a long sequence, e.g., video data. We thus propose our compound memory network, in which each slot stores a series of vectors that are stacked as a matrix representation.

3 Few-shot Video Classification Setup

In the few-shot video classification setting, we aim to train a network that can generalize to new episodes over novel classes. Each episode in a mini-batch mimics a few-shot classification task, which consists of a support set and a query set. The support set contains training videos and labels, while the query set is for evaluating the generalization performance. In an n-way, k-shot problem, the goal of each episode is to classify query videos into n classes with only a small number of support examples per class (k). Videos and labels in an episode are sampled from a meta set. The meta set has N classes ($N > n$), and each class has K examples ($K > k$). In our setup, there are three meta sets, i.e., meta-training set, meta-validation set and meta-testing set with $N_{training}$, $N_{validation}$ and $N_{testing}$ classes, respectively. The meta-training set is for meta-learning which minimizes the loss over training episodes. The meta-validation set is for hyper-parameter tuning. We report the accuracy on the meta-testing set. The three meta sets do not have overlapping categories. Following [23, 32], we construct an episode by randomly choosing n classes from N categories in the meta set. For each class, k videos are selected from K examples. The label indices for n classes are randomly shuffled across different episodes, which prevents the model from memorizing the association between the input and the label.

In a standard video classification problem, there is a single training dataset D_{single} with fixed categories. Given an input/output pair ($\mathbf{x}$, y) sampled from D_{single}, the goal is to minimize the estimated loss over all training examples, i.e.,

$$\min_{\theta} \mathbb{E}_{(\mathbf{x},y)\sim D_{single}}[\mathcal{L}(\mathbf{x}, y)], \tag{1}$$

where θ represents the trainable parameters in a model.

In the few-shot video classification problem, training is conducted over a number of different episodes. An episode T_i sampled from meta-set $\mathcal{T}$ involves an episode length l, inputs $\mathbf{x}_t$, outputs y_t and a loss function $\mathcal{L}(x_t, y_t)$, where $t = \{1, 2, \ldots, l\}$. During meta-training, the network is trained to predict the label of $\mathbf{x}_t$ at each step given previous input pairs $\{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_{t-1}, y_{t-1})\}$. The objective is to minimize the expected loss over mini-batches of episodes, i.e., $\min_\theta \mathbb{E}_{Ti\sim\mathcal{T}}[\sum_{t=1}^{l} \mathcal{L}_i(\mathbf{x}_t, y_t)]$.

4 Compound Memory Network

We first illustrate the multi-saliency embedding function that learns a fixed-size matrix representation for a variable-length video sequence. We then show the detailed structure of our Compound Memory Network, and introduce the

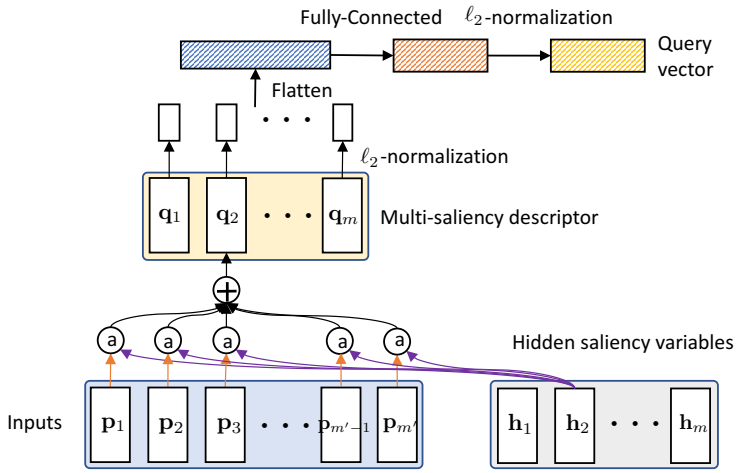

Fig. 2. Illustration of the input embedding model. The embedding function generates the multi-saliency descriptor $\mathbf{Q}$, which is flattened and normalized to a query vector.

novel components, i.e., the constituent keys, abstract memory, together with the accessing and updating operations.

4.1 Multi-saliency Embedding Function

Videos have variable lengths and should be encoded into a fixed-size matrix before being stored in memory. Given a query video $\mathbf{P} = \{\mathbf{p}_1, \mathbf{p}_2, \ldots, \mathbf{p}_{m'}\}$, where m' is the number of video frames and $\mathbf{p}_i$ is a frame-level representation extracted by a ConvNet, video $\mathbf{P}$ should be aggregated into a fixed-size matrix $\mathbf{Q}$. The representation $\mathbf{Q}$ consists of m stacked hidden descriptors $\{\mathbf{q}_1, \mathbf{q}_2, \ldots, \mathbf{q}_m\}$, and the size of each hidden descriptor is `hiden-size`. Note that the number of video frames m' varies across different videos, but m is a fixed number.

We design the multi-saliency embedding function (`MEF`) by introducing a hidden variable $\mathbf{H}$ with m components $\{\mathbf{h}_1, \mathbf{h}_2, \ldots, \mathbf{h}_m\}$. Each component $\mathbf{h}_j$ is used to detect one saliency in a video. For each input $\mathbf{p}_i$, a soft weight $\mathbf{a}_{ij}$ over $\mathbf{h}_j$ will be calculated which measures the relevance between the input and the component. The hidden descriptor $\mathbf{q}_j$ will be the weighted sum over the residual between $\mathbf{P}$ and $\mathbf{h}_j$. Thus, the `MEF` function can be formulated by

$$\mathbf{a}_i = \texttt{softmax}(\frac{\mathbf{p}_i \mathbf{H}^T}{\sqrt{d_{\texttt{hidden-size}}}}), \quad \mathbf{q}_j = \sum_{i=1}^{m} \mathbf{a}_{ij}(\mathbf{p}_i - \mathbf{h}_j), \tag{2}$$

where `softmax` is defined as, $\texttt{softmax}(\mathbf{e}) = \frac{\exp(e_i)}{\sum_i \exp(e_i)}$.

To calculate the relevance score between $\mathbf{p}_i$ and $\mathbf{h}_j$, we simply use dot-product but include a scaled factor $\frac{1}{\sqrt{d_{\texttt{hidden-size}}}}$ [31] followed by a `softmax` function. The original sequence $\mathbf{P}$ is mapped to our multi-saliency descriptor $\mathbf{Q}$, *i.e.*, $\mathbf{Q} =$

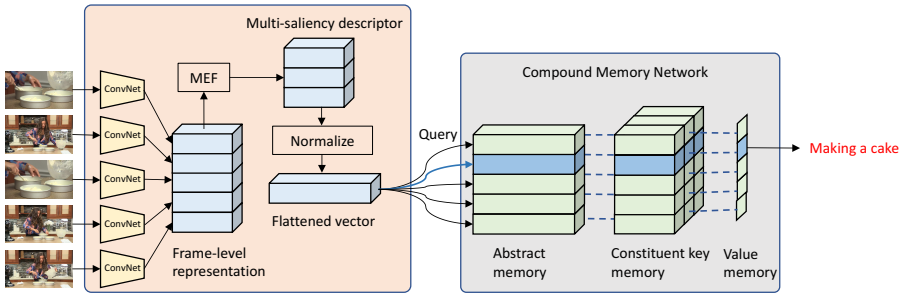

Fig. 3. Our CMN structure. A video is first mapped to a matrix representation via the multi-saliency embedding function. This hidden representation is then vectorized and normalized as a query vector, which performs a nearest neighbour search over the abstract memory. The most similar memory slot is retrieved and the label stored in the value memory will be used as the prediction. The constituent key memory contains the matrix representations of the inputs, while the abstract memory is constructed on top of the stacked constituent keys.

$\texttt{MEF}(\mathbf{P}, \mathbf{H})$. $\mathbf{Q}$ is then flattened and normalized to a vector, which will be discussed in Section 4 (Figure 2). [18, 31] introduced multi-hops attention to calculate multiple weighted sums over the inputs. In contrast, we introduce a hidden variable $\mathbf{H}$ to explicitly model the relation between the input and each hidden vector, which learns multiple descriptors for different salient parts in a video.

4.2 Compound Memory Structure

Our Compound Memory Network is a variant of the Key-Value Memory Networks, which has the key memory ($\mathcal{K}$) and the value memory ($\mathcal{V}$). Visual information is stored in the key part, while the label information is stored in the value part. Our key memory is a layered structure in which the first layer stores the constituent keys ($\mathcal{C}$) and the second layer is the abstract memory ($\mathcal{A}$). We also track the usage of each slot with an age memory ($\mathcal{U}$). Thus, the compound memory module ($\mathcal{M}$) can be represented by the following tuple,

$$\mathcal{M} = ((\mathcal{C}_{\texttt{ns}\times\texttt{nc}\times\texttt{cs}},\ \mathcal{A}_{\texttt{ns}\times\texttt{as}}), \mathcal{V}_{\texttt{ns}},\ \mathcal{U}_{\texttt{ns}}), \tag{3}$$

where $\texttt{ns}$ is the memory size, $\texttt{nc}$ is the number of constituent keys, $\texttt{cs}$ is the key size and $\texttt{as}$ is the abstract memory size.

Two-layer Key Memory In the constituent key memory, we use multiple stacked constituent keys, which have stronger capability than a single vector, as the visual representation. In CMN, each constituent key is represented by a multi-saliency descriptor.

Note that $\mathbf{Q}$ is a matrix with shape (m, $\texttt{hidden-size}$) and there are $\texttt{nc}$ keys in each slot of the constituent key memory. We let m be equal to $\texttt{nc}$, thus each descriptor in $\mathbf{Q}$ can be directly saved in the constituent key memory.

To enable fast nearest neighbour query, we introduce an abstract memory on top of the constituent key memory. The stacked keys are compressed to a vector and it is cached in the abstract memory. The abstract memory can be seen as a snapshot of the constituent key memory. The two sub-memory modules have the same number of slots, but they represent information at different levels.

We denote the stacked matrix representation in $\mathcal{C}$ as $\mathbf{C}$, and each constituent key is $\mathbf{c}_i$, $i \in \{1, \ldots, \mathtt{nc}\}$. We first normalize each constituent key with ℓ_2 normalization, *i.e.*, $\|\mathbf{c}_i\| = 1$. We then flatten the normalized $\mathbf{C}'$ to a vector followed by a Fully-Connected (`FC`) layer, which is then ℓ_2-normalized to a compressed representation. We denote the procedure as the `normalize` function,

$$\mathbf{c}_i{}' = \frac{\mathbf{c}_i}{\|\mathbf{c}_i\|}, \quad \mathbf{d}' = \mathtt{FC}(\mathtt{flatten}(\mathbf{C}')), \quad \mathbf{d} = \frac{\mathbf{d}'}{\|\mathbf{d}'\|}, \tag{4}$$

where a `FC` layer is simply a linear transformation layer, *i.e.*, $\mathtt{FC}(\mathbf{x}) = \mathbf{w}\mathbf{x}+b$. The compressed representation $\mathbf{d}$ is stored in the abstract memory, which will only be updated when the value in constituent key memory changes. The abstract memory keeps a one-to-one mapping to the constituent key memory, which will accelerate the query process.

Reading Given a query vector $\mathbf{z} = \mathtt{normalize}(\mathbf{Q})$, nearest neighbour search is conducted over the abstract memory. We select the memory slot that is closest to the query $\mathbf{z}$ by, $\mathtt{NN}(\mathbf{z},\ \mathcal{A}) = \mathrm{argmax}_i\ \mathbf{z} \cdot \mathcal{A}[i]$. k-nearest slots (ordered by decreasing similarity) can be returned by,

$$(n_1, \ldots, n_k) = \mathtt{NN}_k(\mathbf{z},\ \mathcal{A}), \tag{5}$$

where n_1 is the memory slot that is most similar to the query. At the inference phase, $\mathcal{V}[n_1]$ will be our prediction for query $\mathbf{z}$.

Writing The new information should be recorded in the memory to reflect the relation of new query $\mathbf{z}$ and the corresponding label y. The memory will not be updated via backpropagation which may catastrophically modify the information, but it will be refreshed with the following rule. Note that n_1 is the index of the nearest memory slot, and if the memory already returns the correct label, *i.e.*, $\mathcal{V}[n_1] = y$, we only update the n_1 memory slot. $\mathcal{A}[n_1], \mathcal{U}[n_1]$ and $\mathcal{C}[n_1]$ will be updated, and leave $\mathcal{V}[n_1]$ unchanged.

$$\begin{aligned} \mathcal{C}[n_1][i] &\leftarrow \mathbf{q}_i + \mathcal{C}[n_1][i], \quad \text{for } i = 1, \ldots \mathtt{nc}, \\ \mathcal{A}[n_1] &\leftarrow \mathtt{normalize}(\mathcal{C}[n_1]), \quad \mathcal{U}[n_1] \leftarrow 0. \end{aligned} \tag{6}$$

The constituent key memory is updated by averaging the constituent keys $\mathcal{C}[n_1]$ and the multi-saliency descriptors $\mathbf{Q}$. The abstract memory $\mathcal{A}[n_1]$ is updated correspondingly. We also set $\mathcal{U}[n_1]$ to 0, which shows that slot n_1 has just been updated.

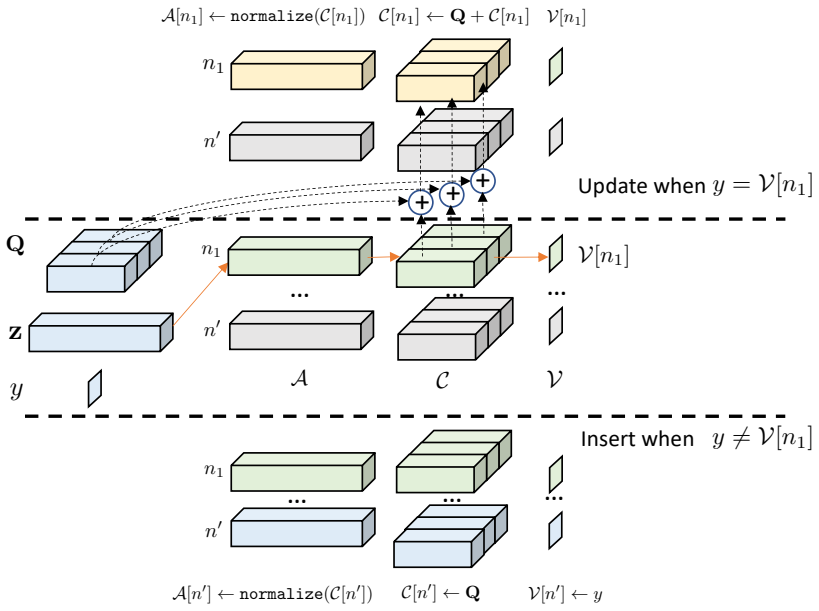

Fig. 4. Illustration of the update rule for CMN.

When $\mathcal{V}[n_1] \neq y$, the $(\mathbf{Q}, y)$ pair is stored in another memory slot to record the information. We choose the oldest memory slot n' that has not been updated for a long time,

$$n' = \arg\max_i (\mathcal{U}[i] + r_i), \tag{7}$$

where r_i is a random number to introduce randomness during slot selection. The memory will be updated by,

$$\begin{aligned} \mathcal{C}[n'][i] &\leftarrow \mathbf{q}_i, \quad \text{for } i = 1, \ldots, \texttt{nc}, \\ \mathcal{A}[n'] \leftarrow \texttt{normalize}(\mathcal{C}[n']), \quad \mathcal{V}[n'] &\leftarrow y, \quad \mathcal{U}[n'] \leftarrow 0. \end{aligned} \tag{8}$$

In this case, $\mathcal{V}[n']$ is also updated with the new label y. We illustrate the procedure in Figure. 4.

4.3 Training

Given a query $\mathbf{z}$ and a corresponding ground-truth label y, we retrieve top-k key-value pairs on memory indices $(n_1, \ldots, n_k)$ by Eq. 5. Let `i-pos` be the smallest index that $\mathcal{V}[n_{\texttt{i-pos}}] = y$ and `i-neg` be the smallest index that $\mathcal{V}[n_{\texttt{i-neg}}] \neq y$. We train the query vector $\mathbf{z}$ to be more similar to $\mathcal{A}[n_{\texttt{i-pos}}]$ than $\mathcal{A}[n_{\texttt{i-neg}}]$ with the following ranking loss,

$$\mathcal{L}(\mathbf{z}, y, \mathcal{A}) = \max(\alpha - \mathbf{z} \cdot \mathcal{A}[n_{\texttt{i-pos}}] + \mathbf{z} \cdot \mathcal{A}[n_{\texttt{i-neg}}], 0). \tag{9}$$

The similarity between the query and the positive key should be larger than the similarity between the query and the negative key by margin α. The loss will be 0 when the difference between the two similarities is beyond margin α.

The memory in each episode is cleared before operations are conducted. The clear operation simply initializes all memory variables to 0. During mini-batch training, information from multiple episodes are stored in the global memory. To avoid confliction in the label space, label ids across episodes should be different. The global label id can be calculated by,

$$\texttt{global-label-id} = \texttt{label-id} + \texttt{index} \times \texttt{k}, \tag{10}$$

where `k` is the number of classes, `label-id` is the shuffled label id in an episode and `index` is the index of the episode in the mini-batch. At the inference phase, the weights of the network are fixed except for the memory module, which will be updated with the support set examples.

5 Experiments

5.1 Datasets

There are no existing datasets for few-shot video classification, thus we collected the first dataset for few-shot video classification evaluation, which we will release for future research. We used videos from the recently released Kinetics dataset [13], which consists of 400 categories and 306,245 videos, covering videos from a wide range of actions and events, e.g., "dribbling basketball", "robot dancing", "shaking hands", "playing violin". We randomly selected 100 classes from the Kinetics dataset, each of which contains 100 examples. The 100 classes were split into 64, 12 and 24 non-overlapping classes for use as the meta-training set, meta-validation set and meta-testing set, respectively.

5.2 Implementation Details

In an n-way, k-shot problem, we randomly sampled n classes and each class has k examples, while an additional unlabeled example belonging to one of the n classes is used for testing. Thus each episode has $nk+1$ examples. We calculated the mean accuracy by randomly sampling 20,000 episodes in all experiments.

To obtain the frame-level feature representation, we forwarded each frame to a ResNet-50 [9] network that was pre-trained on ImageNet. We followed the basic image preprocessing procedure, whereby the image was first rescaled by resizing the short side to 256 and a 224×224 region was randomly cropped from the image. We cropped the central region during the inference phase.

We optimized our model with Adam [14] and fixed the learning rate to 1.0×10^{-4}. The margin α was set to 0.5 in all experiments. We tuned the hyper-parameters on the meta-validation set, and stopped the training process when the accuracy on the meta-validation set began to decrease. The model was implemented with the TensorFlow framework [1].

Table 1. Results of 5-way few-shot video classification on the meta-testing set. The numbers are reported in percentages. Our CMN achieves state-of-the-art results.

Model	1-shot	2-shot	3-shot	4-shot	5-shot
RGB w/o mem	28.7	36.8	42.6	46.2	48.6
Flow w/o mem	24.4	27.3	29.8	32.0	33.1
LSTM(RGB) w/o mem	28.9	37.5	43.3	47.1	49.0
Nearest-finetune	48.2	55.5	59.1	61.0	62.6
Nearest-pretrain	51.1	60.4	64.8	67.1	68.9
MatchingNet [32]	53.3	64.3	69.2	71.8	74.6
MAML [5]	54.2	65.5	70.0	72.1	75.3
Plain CMN [11]	57.3	67.5	72.5	74.7	76.0
LSTM-emb	57.6	67.9	72.8	74.8	76.2
Ours	**60.5**	**70.0**	**75.6**	**77.3**	**78.9**

5.3 Evaluation

We compare our model with several baselines. We report 1-shot, 2-shot, 3-shot, 4-shot and 5-shot results on the 5-way classification task. In the first baseline, we utilize all the training data to pre-train the ResNet-50 network. At the testing stage, we fine-tune the network for each episode. The network is initialized with the pre-trained weights up to the last layer. The weights in the last layer is randomly initialized. We test the performance with different inputs. For "RGB w/o mem", we take RGB frames as inputs to train the network. For "Flow w/o mem", stack flows images are stacked as inputs to the network. To encode video with more sophisticated embedding function upon the frame-level features, we use an LSTM to aggregate temporal dynamics in a video. The LSTM takes the RGB features as inputs. It is fine-tuned for each episode. We denote this baseline as "LSTM (RGB) w/o mem". Another baseline is a nearest neighbour baseline ("Nearest-finetune"). We first finetune the ResNet-50 network to classify all classes in the meta-training set. We feed each frame as the input image and the video-level label is used as the label for each frame. Frames are first preprocessed with the procedure described above. We initialize the weights of the ResNet-50 network with the ImageNet pre-trained model. We train the network via stochastic gradient descent (SGD) with momentum 0.9. We set the initial learning rate to 0.01. We decrease the learning rate by 0.1 every 10 epochs. The batch size is 128. During inference, we feed the video frames to the finetuned ResNet-50 network and extract the activations from the last layer before final classification. We average the frame-level features and obtain a video-level representation of 2,048 dimension. We also apply ℓ_2 normalization before nearest neighbour search.

In the next baseline ("Nearest-pretrain"), we do not finetune the ResNet-50 network on the meta-training dataset, but directly utilize the pre-trained weights without modification. We embed the video with the same procedure in "Nearest-finetune", and then apply nearest neighbour search.

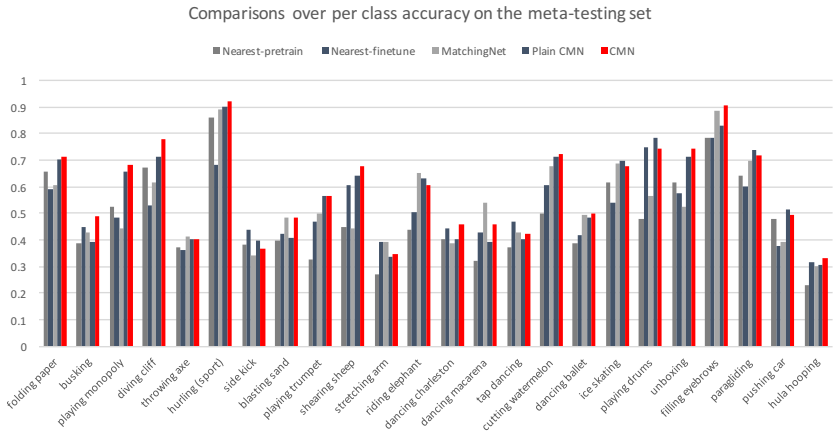

Fig. 5. Per class accuracy on the 5-way 1-shot setting. We show the accuracies of 24 classes on the meta-testing set.

We also show the result of the Matching Network [32] ("MatchingNet") on this dataset, which achieves state-of-the-art performance on the few-shot image classification task. We implement the Matching Network algorithms ourselves. We first feed the frames to a ResNet-50 network without fine-tuning. We average frame-level features to obtain a video-level feature. We then use the fully-conditional embedding (FCE) function proposed in [32] to embed the training examples. The FCE uses a bidirectional-LSTM and each training example is a function of all the other examples. To train MAML [5], we average the frame-level features and follow the default hyper-parameters in [5].

Another baseline is "Plain CMN" where we remove the constituent key memory from the model and use a video-level vector as video representation. We replace our embedding module with an LSTM function, while keeping the other settings the same. We denote this baseline as "LSTM-emb". We conduct this baseline to show the effectiveness of our compound memory network structure.

The results are shown in Table 1. We can see from Table 1 that our CMN improves the baselines in all shots. We observe that fine-tuning the ResNet-50 network on the meta-training set does not improve the few-shot video classification performance, but significantly harms performance. As there are no overlapping classes between the meta-training set and the meta-testing set, it is very likely that the model will overfit the meta-training set. Our CMN structure also outperforms the Matching Networks by more than 4% across all shots. Furthermore, our CMN structure outperforms the "Plain CMN", which demonstrates the strong representation capability of the constituent key memory. About 10% improvement is obtained between the 1-shot setting and the 2-shot setting, by only adding one example per class. The relative improvement decreases when more examples are added, e.g., the improvement from 3-shot to 4-shot is only

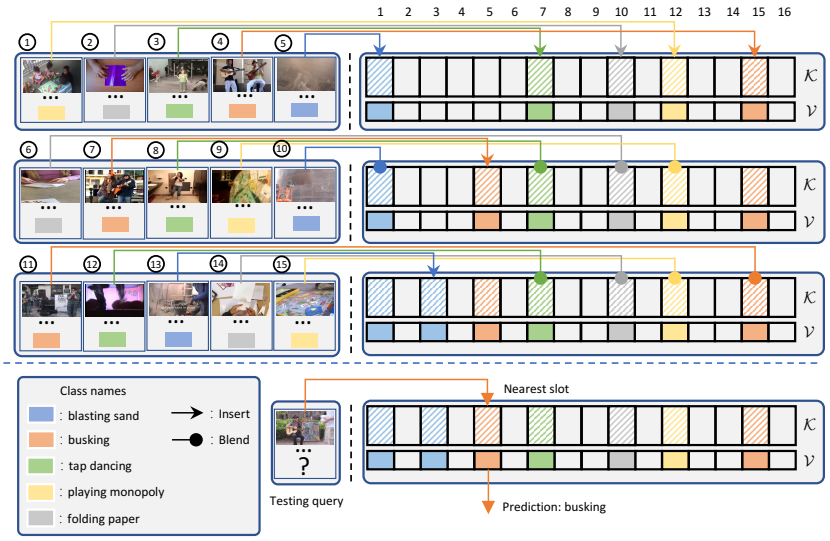

Fig. 6. We illustrate the inference procedure. There are 5 classes and the memory has 16 slots. Two different update rules will be used depending on the query results.

Table 2. Results of different memory sizes.

Model	1-shot	2-shot	3-shot	4-shot	5-shot
Mem-64	52.0	61.9	66.5	69.4	71.2
Mem-128	53.4	63.7	68.9	71.5	73.5
Mem-512	**55.1**	**65.3**	**70.1**	72.0	**74.2**
Mem-2048	55.0	65.0	69.7	**72.4**	74.1

1.7%. This shows that one-shot classification is still a difficult problem which can be further improved in the future.

The 1-shot accuracy for each class is shown in Figure 5. We report the mean accuracy for class c over all episodes where the query label is c. The "hurling (sport)" category have the highest accuracy, while "hula hooping" and "stretching arms" achieve the worst performance with about 30% accuracy.

We illustrate the inference procedure in an episode in Figure 6. In this 5-way 3-shot setting, the support set has 15 examples. Each example is sequentially fed to the network. This episode is divided into three groups, each of which has five examples with distinct labels. We arrange the episode in this way for better illustration. In row 1, all inputs are inserted into the memory. In row 2, the 7th example is inserted into a new slot in the memory, while other videos are blended into existing slots of the same category. In row 3, the 13th example is inserted. For the 11th example, the closest slot is the 15th slot and the two representations are averaged.

Table 3. Results of different numbers of multi-saliency descriptors.

Model	1-shot	2-shot	3-shot	4-shot	5-shot
Desc-1	53.7	63.5	68.3	70.9	73.3
Desc-5	**55.1**	**65.3**	**70.1**	**72.0**	**74.2**
Desc-10	53.2	62.9	68.2	70.0	72.3

Table 4. Results of different way few-shot video classification.

Model	1-shot	2-shot	3-shot	4-shot	5-shot
5-way	55.0	65.0	69.7	72.4	74.1
6-way	51.7	61.8	66.4	69.3	71.2
7-way	49.5	59.6	64.3	67.1	68.9
8-way	46.0	56.1	61.0	64.0	65.8

5.4 Ablation Study

We perform ablation experiments to explain our selections for the final model. The default setting is the 5-way few-shot classification. We show the performance of different memory sizes in Table 2, and the results of different numbers of constituent keys are shown in Table 3. We also report the results of other few-shot video classification tasks with different numbers of categories. We report the results on the meta-validation set, and choose only 10 frames during evaluation.
Memory size. The results of different memory sizes are shown in Table 2. When the memory has a small number of slots, the performance is worse because some information has to be wiped out when new data arrives. Memory size of 512 achieves the best results. Increasing the memory size does not improve performance when the memory is large enough to record all the information.
The number of multi-saliency descriptors. The result is shown in Table 3. It shows that multi-saliency descriptors with stronger representation capability obtain better performance than a single descriptor. The performance decreases when too many descriptors are used, because more parameters are introduced in the network.
***N*-way classification**. In all previous experiments, evaluations were conducted on the 5-way classification setting. n-way classification with larger n is a similar task to 5-way classification, but can be more difficult. As can be seen, the performance decreases when n increases.

6 Conclusion

In this paper, we have proposed a compound memory network for few-shot video classification. This module stores matrix representations, which can be easily retrieved and updated in an efficient way. Our future work is to leverage multiple memory banks of different modality representations.
Acknowledgment. Our work is partially supported by the Data to Decisions CRC (D2D CRC) and the Cooperative Research Centres Programme. We gratefully acknowledge the support of NVIDIA Corporation with the donation of the TITAN X (Pascal) GPU. We thank AWS Cloud Credits for Research for supporting this research.

References

1. Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., Isard, M., Kudlur, M., Levenberg, J., Monga, R., Moore, S., Murray, D.G., Steiner, B., Tucker, P., Vasudevan, V., Warden, P., Wicke, M., Yu, Y., Zheng, X.: Tensorflow: A system for large-scale machine learning. In: OSDI (2016)
2. Bahdanau, D., Cho, K., Bengio, Y.: Neural machine translation by jointly learning to align and translate. In: ICLR (2015)
3. Carreira, J., Zisserman, A.: Quo vadis, action recognition? a new model and the kinetics dataset. In: CVPR (2017)
4. Fei-Fei, L., Fergus, R., Perona, P.: One-shot learning of object categories. TPAMI **28**(4), 594–611 (2006)
5. Finn, C., Abbeel, P., Levine, S.: Model-agnostic meta-learning for fast adaptation of deep networks. In: ICML (2017)
6. Graves, A., Wayne, G., Danihelka, I.: Neural Turing machines. arXiv preprint arXiv:1410.5401 (2014)
7. Hariharan, B., Girshick, R.: Low-shot visual recognition by shrinking and hallucinating features. In: ICCV (2017)
8. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: CVPR (2016)
9. He, K., Zhang, X., Ren, S., Sun, J.: Identity mappings in deep residual networks. In: ECCV (2016)
10. Hochreiter, S., Schmidhuber, J.: Long short-term memory. Neural Computation **9**(8), 1735–1780 (1997)
11. Kaiser, Ł., Nachum, O., Roy, A., Bengio, S.: Learning to remember rare events. In: ICLR (2017)
12. Karpathy, A., Toderici, G., Shetty, S., Leung, T., Sukthankar, R., Fei-Fei, L.: Large-scale video classification with convolutional neural networks. In: CVPR (2014)
13. Kay, W., Carreira, J., Simonyan, K., Zhang, B., Hillier, C., Vijayanarasimhan, S., Viola, F., Green, T., Back, T., Natsev, P., et al.: The kinetics human action video dataset. arXiv preprint arXiv:1705.06950 (2017)
14. Kingma, D., Ba, J.: Adam: A method for stochastic optimization. In: ICLR (2015)
15. Koch, G.: Siamese Neural Networks for One-Shot Image Recognition. Ph.D. thesis, University of Toronto (2015)
16. Krizhevsky, A., Sutskever, I., Hinton, G.E.: ImageNet classification with deep convolutional neural networks. In: NIPS (2012)
17. Lake, B., Salakhutdinov, R., Gross, J., Tenenbaum, J.: One shot learning of simple visual concepts. In: CogSci (2011)
18. Lin, Z., Feng, M., Santos, C.N.d., Yu, M., Xiang, B., Zhou, B., Bengio, Y.: A structured self-attentive sentence embedding. In: ICLR (2017)
19. Miller, A., Fisch, A., Dodge, J., Karimi, A.H., Bordes, A., Weston, J.: Key-value memory networks for directly reading documents. In: EMNLP (2016)
20. Miller, E.G., Matsakis, N.E., Viola, P.A.: Learning from one example through shared densities on transforms. In: CVPR (2000)
21. Ravi, S., Larochelle, H.: Optimization as a model for few-shot learning. In: ICLR (2017)
22. Ren, S., He, K., Girshick, R., Sun, J.: Faster R-CNN: Towards real-time object detection with region proposal networks. In: NIPS (2015)

23. Santoro, A., Bartunov, S., Botvinick, M., Wierstra, D., Lillicrap, T.: Meta-learning with memory-augmented neural networks. In: ICML (2016)
24. Simonyan, K., Zisserman, A.: Two-stream convolutional networks for action recognition in videos. In: NIPS (2014)
25. Simonyan, K., Zisserman, A.: Very deep convolutional networks for large-scale image recognition. In: ICLR (2015)
26. Snell, J., Swersky, K., Zemel, R.S.: Prototypical networks for few-shot learning. In: NIPS (2017)
27. Sukhbaatar, S., Weston, J., Fergus, R., et al.: End-to-end memory networks. In: NIPS (2015)
28. Sutskever, I., Vinyals, O., Le, Q.V.: Sequence to sequence learning with neural networks. In: NIPS (2014)
29. Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., Wojna, Z.: Rethinking the inception architecture for computer vision. In: CVPR (2016)
30. Tran, D., Bourdev, L., Fergus, R., Torresani, L., Paluri, M.: Learning spatiotemporal features with 3d convolutional networks. In: ICCV (2015)
31. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, L., Polosukhin, I.: Attention is all you need. In: NIPS (2017)
32. Vinyals, O., Blundell, C., Lillicrap, T., Wierstra, D., et al.: Matching networks for one shot learning. In: NIPS (2016)
33. Wang, H., Schmid, C.: Action recognition with improved trajectories. In: ICCV (2013)
34. Wang, L., Xiong, Y., Wang, Z., Qiao, Y., Lin, D., Tang, X., Van Gool, L.: Temporal segment networks: Towards good practices for deep action recognition. In: ECCV (2016)
35. Weston, J., Chopra, S., Bordes, A.: Memory networks. In: ICLR (2015)
36. Wu, Y., Schuster, M., Chen, Z., Le, Q.V., Norouzi, M., Macherey, W., Krikun, M., Cao, Y., Gao, Q., Macherey, K., et al.: Google's neural machine translation system: Bridging the gap between human and machine translation. arXiv preprint arXiv:1609.08144 (2016)
37. Xu, Z., Zhu, L., Yang, Y.: Few-shot object recognition from machine-labeled web images. In: CVPR (2017)
38. Yue-Hei Ng, J., Hausknecht, M., Vijayanarasimhan, S., Vinyals, O., Monga, R., Toderici, G.: Beyond short snippets: Deep networks for video classification. In: CVPR (2015)
39. Zhu, L., Xu, Z., Yang, Y.: Bidirectional multirate reconstruction for temporal modeling in videos. In: CVPR (2017)