k-Shot Learning for Action Recognition

Darshan Thaker *
The University of Texas at Austin
Department of Computer Science
dbthaker@cs.utexas.edu

Kapil Krishnakumar *
The University of Texas at Austin
Department of Computer Science
krishnakumar.kapil@utexas.edu

Abstract

In the problem of k-shot learning, a model must learn to reliably classify an example having seen only k previous instances of examples of the same class. With recent success in using memory in neural networks to perform k-shot learning, we propose a technique that uses Memory-Augmented Neural Networks to perform k-shot learning for action recognition in videos. We believe the use of memory will help learn encoded generalizable representations of actions that can be used for classification. We use the Memory-Augmented Neural Network framework to evaluate k-shot learning for the Kinetics Dataset of action clips taken from YouTube videos. Our approach works well for this domain, achieving results close to 78% accuracy in the 9-shot learning case.

1. Introduction

The general problem of video understanding deals with enabling computers to *understand* what is occurring in a video scene, whether it be the simple phenomenon of movement of certain objects to predicting actions in video. With the prominence of video in social media platforms such as Facebook, Google, YouTube, etc., understanding video is a domain extremely vital to the goal of computers reaching human-levels of visual recognition. *Action recognition* deals with the problem of predicting actions being performed in a video. To successfully approach action recognition, a computer must have a holistic understanding of the video to track the actor in a video, any objects the actor might be acting upon, and how their relationship is manifested. For example, for the action 'mowing a lawn', a computer would need to correctly identify the scene in a video as a lawn, identify an actor as holding a lawnmower, and realize that moving the lawnmower across the lawn connotes the action of 'mowing a lawn'.

Current state-of-the-art techniques require thousands of prior examples of specific actions to achieve competitive action recognition accuracy levels [7, 4, 2, 22, 11]. In contrast, humans can correlate actions with very few prior examples. A large facet of achieving human-level visual recognition performance is also the ability to discriminate between actions and scenes with few previous examples. The problem of learning from few examples is coined as few-shot learning, or *k-shot learning*. Specifically, for a k-shot learning setup, a model must learn an example-class binding from only k examples of a class, where k is relatively small $(1 - 10)$. We tackle the problem of k-shot learning for action recognition. This has numerous applications, from Virtual Reality/Augmented Reality systems to robotics. For instance, a human might want to customize certain gestures/actions in a Virtual Reality system to bind gestures with tasks. Additionally, one can imagine a robot learning to perform an action from few demonstrations of that action. This problem is very challenging because a model must learn a generalizable representation of an action from very few examples in the input space.

Contributions. The premise of our work is the hypothesis that humans can tackle k-shot action recognition well because of the use of memory. Memory plays a key role in serializing humans' experiences, and we believe the use of memory is integral to a successful method for performing k-shot action recognition. To that end, we propose the use of *Memory-Augmented Neural Networks* [18] to tackle k-shot action recognition. A Memory-Augmented Neural Network (MANN) consists of a controller and an external memory module, which can be used for storing "memories" corresponding to an action. We experiment with representing the video input as a *dynamic image* [2], and introduce a convolutional encoder in the controller portion of the MANN. With this setup, we find that for a 5-way classification task for action recognition, MANNs achieve 78% accuracy in the 9-shot learning task, and perform significantly better than a naive baseline and random guessing in the 5 to 8-shot learning task as well.

*Indicates equal contribution.

2. Related Work

2.1. Action Recognition

The problem of recognizing actions in videos has been a hot area of research over the last few years. For the most part, research within the last couple of years in action recognition can be split up into two areas: *choice of video representation* and *choice of a neural network architecture*.

The first category of video representations chooses to represent a video as a static bag of frames [22, 23]. This has been shown to give good results for action recognition [23], since an action can sometimes be identified in a static frame by certain visual cues such as background or lighting. This naive approach falls short in fine-grained action recognition challenges, which led to improvements upon this representation, one of which includes coupling the static RGB frames with their corresponding optical flow fields [7]. This representation helps expose spatio-temporal connections between frames that are not visible from a static RGB frame (e.g. difference between opening and closing a door). More advanced features can also be extracted from video by treating videos as groups of supervoxels and finding segments of video frames that are most discriminative of the action being performed in the video [9]. Neural network architectures to tackle action recognition have usually revolved around using deep convolutional neural networks with the various video representations described above. Some of these architectures include 3-D convolutional networks [11, 7], convolutional networks with a recurrent long short-term memory (LSTM) layer [6], or two-stream convolutional architectures that use both image and optical flow data [7].

2.2. One-shot Learning

There have been few previous approaches of one-shot learning, specifically in the video domain. Some of these approaches include the use of one-shot learning techniques for video segmentation [3]. Most of the approaches to one-shot learning test their approaches on the Omniglot dataset [15], which is a dataset that contains 1623 handwritten characters from 50 different alphabets. These approaches include the use of memory-augmented neural networks [18], the use of Siamese networks [13], and matching networks [21]. The Omniglot dataset is a relatively simple and fine-grained dataset, with handwritten characters in black text on a white background, unlike any video dataset with real-world scenes. Thus, we hope our approach shows the versatility of memory-based approaches not only for simpler datasets such as the Omniglot dataset, but for more advanced video datasets as well.

3. Background

3.1. Dynamic Images for Action Recognition

Bilen *et al*. introduced an algorithm to condense a video into a single image[2], giving good results on various action recognition benchmarks. When large video data is presented to a CNN, the designers of the network architecture must keep track of many things, such as the size of the video, mechanisms for sampling frames, as well as architectures that process frames concurrently. Instead, condensing a video into an image harnesses the efficient nature of CNN processing of a still image.

The method of Bilen *et al*. works by learning a ranking function over the frames of the video. Formally, consider the T frames of a video as $I = I_1, I_2, \ldots, I_T$, each of size $m \times n$. Consider the time average of these frames up to a time t as $V_t = \frac{1}{t}\sum_{\tau=1}^{t} I_\tau$. We wish to learn a feature vector d of size $m \cdot n$ that parametrizes a ranking/scoring function $S(t|d) = \langle d, V_t \rangle$. Specifically, we wish to learn the relationship that later frames in the video are assigned higher scores, or that

$$q > t \implies S(q|d) > S(t|d) \tag{1}$$

The intuition behind this method is that if a feature vector d can rank frames successfully in a video, it contains aggregated information from the video and can be used as a feature descriptor for the entire video. Specifically, the *dynamic image* for a video is defined as the $m \cdot n$ feature vector d reshaped into a $m \times n$ image, so it can interpreted as a still image of the same size as a single video frame.

A convex optimization problem is formulated to learn d, similar to the RankSVM formulation [20]. This optimization problem is described in Equation 2. The first term of the optimization is a hinge loss that counts how many pairs of times are misclassified (i.e. two frames such that one precedes the other temporally but is assigned a higher score by the scoring function). This hinge-loss is strengthened a bit by enforcing a margin of size 1 between scores, as usually performed in Support Vector Machine optimization problems. The second term is a quadratic regularizer term.

$$\arg\min_d \frac{2}{T(T-1)} \sum_{q>t} \max\{0, 1 - S(q|d) + S(t|d)\} + \frac{\lambda}{2}\|d\|^2 \tag{2}$$

Dynamic images have been shown to focus primarily on actors and the objects which they act upon in videos, and have been shown to omit noisy background information, as demonstrated in Figure 1. As such, this representation is useful to pass into a CNN as opposed to an entire video since the dynamic image encodes useful condensed information about the video.

Figure 1: Sample dynamic image for action 'drumming'

3.2. Memory in Neural Networks

Some of the first uses of memory in neural networks involve the use of attention mechanisms, which still have prominent use today in visual attention models [17]. With the introduction of the Long Short-Term Memory network (LSTM), neural networks were able to have a more concrete notion of memory embedded within them, and these networks performed well at analyzing various sequential data such as time-series data [10]. Neural Turing Machines first introduced the concept of differentiable external memory to a neural network [8]. The motivation for using external memory as opposed to "internal" memory like an LSTM is twofold. First, it introduces a scalable architecture that can quickly store significant amounts of new information. Secondly, the number of trainable parameters is distinct from the size of memory, unlike an LSTM.

3.3. Memory-Augmented Neural Networks

We focus on the Memory-Augmented Neural Network (MANN) model proposed by Santoro *et al*. [18]. This model consists of two parts: a *controller* and a *memory module*, both of which are described in the subsequent sections.

3.3.1 Controller

The controller in the MANN is the component responsible for generating and retrieving memories that are stored in the external memory module. Santoro *et al*. experimented with 2 types of controllers: LSTM and Feed Forward Networks in their experiments on the Omniglot dataset [15]. The LSTM controller greatly outperformed the pure Feed Forward variation, so we focus on that controller.

The controller takes in as input one image and the previous memory retrieved r_{t-1}. The images in the Omniglot dataset are flattened, though this is not a necessity of the input representation. The output of the controller is a set of keys, which are used to index into the memory module for read and write operations (as described in Section 3.3.2). The auxiliary output that the controller generates is a parameter α used in generating write weights as explained later. For LSTM based controllers, the state of the LSTM is additionally outputted as the recurrent part of the model.

3.3.2 Memory Module

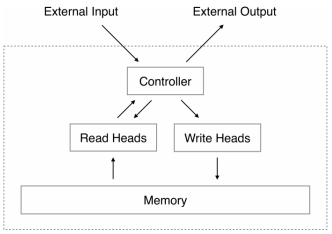


Figure 2: High-level Memory Module Interface (Figure from [8])

The data structure used for the external memory is a matrix M_t of size $m \times n$. The rows of M_t serve as memory 'slots' with the data inside the rows representing individual memories. These rows are written and read to via multiple *write heads* and *read heads*, as shown in Figure 2. Whilst a traditional matrix can be written and accessed to using array indexing, these operations are not differentiable. To make write and read operations differentiable, in the MANN approach, write and access operations must be performed over the whole matrix. This is done using 'blurry' indexing where write and read operations are weighted linear combinations defined over the rows of M_t. Below, we describe in detail how these weights are generated for both write and read operations.

In order to write to the memory, the MANN generates write weights as a linear combination between two forms of write mechanisms: a) a most recently used mechanism (based on read weights w_{t-1}^r from the previous timestep) (b) a least recently used (LRU) mechanism (based on weights w_{t-1}^{lu} from the previous timestep). This is shown in Equation 3. The weight α is generated by the controller and is passed through a sigmoid function $\sigma(\cdot)$.

$$w_t^w \leftarrow \sigma(\alpha) w_{t-1}^r + (1 - \sigma(\alpha)) w_{t-1}^{lu} \tag{3}$$

The LRU weight vector acts as an indicator for the memory slots that need to be overwritten due to lack of usage recently in the sequence. To compute the LRU weights, the system first maintains a usage weight vector w_t^u, that is updated every read and write step defined in Equation 4. In this equation, γ is a fixed scalar tunable hyperparameter that determines the decay of previous usage values (usually set to 0.99). From the usage weight vector w_t^u, the indices of

the smallest $|h|$ (number of read heads) values of w_{t-1}^u are computed. Next, these indices are set to 1 while the rest of the indices of the vector are set to 0, which gives the final least usage weight vector. The least usage weight vector is used to determine which slots can be overwritten, and a linear combination of this weight vector and the read weights from the previous timestep determine which slots need to be updated for the current timestep (i.e. the current write weights).

$$w_t^u \leftarrow \gamma w_{t-1}^u + w_t^r + w_t^w \tag{4}$$

The memory matrix is updated after these weights are generated at every step in the sequence. The rows found to be the least recently used are set to zero. The memory update operation uses the computed write weights in combination to the keys generated from the controller as shown in Equation 5:

$$\forall i: \; M_t(i) \leftarrow M_{t-1}(i) + w_t^w(i)k_t \tag{5}$$

For a given read head r and a key k outputted by the controller, read weights w_t^r are generated based upon the cosine similarity of the key k with the rows of the memory matrix followed by a softmax operation, as shown in Equation 6 and 7.

$$K(k_t, M_t(i)) = \frac{k_t \cdot M_t(i)}{\|k_t\| \cdot \|M_t(i)\|} \tag{6}$$

$$w_t^r(i) = \frac{exp\Big(K(k_t, M_t(i))\Big)}{\sum_j exp\Big(K(k_t, M_t(j))\Big)} \tag{7}$$

Next, the output read vector or memory r_t is generated from the computed read weights and the updated memory matrix based on Equation 8. This read vector is passed back into the controller to perform future memory extractions.

$$r_t \leftarrow \sum_i^R w_t^r(i) M_t(i) \tag{8}$$

The read vector is used by the system as input into a classifier, a softmax output layer, to make the class prediction step for that particular image in the sequence.

3.3.3 Input Representation

Unlike conventional neural networks, where input data is processed independently per image in the training data, the input data for a MANN is processed in *sequences*. A sequence is a list of 2-tuples storing an input image (x_t) and a temporally-shifted label (y_{t-1}). The temporally-shifted labels are shifted by 1 time-step into the future, so the true label y_t for an input image x_t is associated with the input image x_{t+1} in the sequence. For example, a sample sequence of length T might be

$$(x_1, null), (x_2, y_1), (x_3, y_2), \ldots, (x_T, y_{T-1})$$

A MANN iteratively processes every 2-tuple in a given sequence by taking in as input an image x_t and a shifted label y_{t-1}. The network is tasked with predicting the true label y_t, even though it has not yet seen the true label. The intuition behind this setup is that during training, the network should learn to emit read vectors that are similar to examples with the same true label as the input image. The true label for a given image will be presented in the following example in the sequence, which means the network must write a representation of the input image into memory first. Ideal performance would involve a random guess by the MANN when the first example of a given label is seen, and when a second example for that label occurs in the sequence, the MANN should emit a read vector that is discriminative enough to allow the network to predict the correct label using its memory. In other words, network will hold data samples in memory until the correct label is presented (one timestep later), at which point sample-class bindings can be learned.

3.3.4 Training

Sequences are grouped into mini-batches that are used for training and testing. The loss function used for classification is the cross-entropy loss between the network output and the one-hot labels. The network attempts to minimize the loss between the output classification and true labels based on:

$$\mathcal{L}(\theta) = -\sum_t \langle y_t, \log p_t \rangle \tag{9}$$

where p_t is the output probabilities outputted by the MANN and y^t is the true unshifted label expected at time t. The trainable parameters in the network are the LSTM gates, the encoder, and the softmax classifier.

3.3.5 Evaluation and Accuracy Values

The evaluation is performed by generating a sequence from the test label set. The sequence is fed into the MANN using the shifted label approach described in Section 3.3.3 and the output is a sequence of predicted labels for every point in the sequence. Accuracy bins are created that are indexed by the number of times a class was seen before the current instance in the sequence. For example, 1-shot learning would focus on the case where only 1 example of a given label is seen before encountering another example of the same label. For every point in the sequence, we keep a count

of previous examples of that class in the sequence and update the value of the associated k-shot accuracy bin based on the prediction in comparison to the true label. For example, for the label sequence $[1, 3, 2, 1, 2, 2, 3]$ and output sequence $[1, 2, 2, 1, 1, 2, 3]$, the resulting accuracy bins for 0-shot, 1-shot, and 2-shot are $0.66, 0.66, 1$ respectively.

4. Approach

For our dataset, we use the Kinetics dataset [12]. The Kinetics dataset contains 400 action classes, with more than 400 YouTube ($\approx$ 10 second) video clips for each action. We sample a subset of the $\approx$ 400 videos from each action for scalability reasons and perform training and testing with a 50-50 disjoint split on the action labels. We chose this dataset because of the large number of unique action classes that allowed us to test k-shot recognition on many novel held-out actions that are not in the training set.

Every video in the dataset is condensed down to a $64 \times 64 \times 3$ dynamic image, as described in Section 3.1 [2]. This was chosen for two reasons. First, it significantly speeds up training time since each forward pass through the network only involves operations on a single still image as opposed to a video. Secondly, the dynamic image reasonably captures actors and objects upon which they act in a video, which is a good way to filter background noise and other irrelevant information in a video. This is especially important in the k-shot learning task through the usage of memory, since we do not want a memory-augmented neural network to store in memory a representation of the video that is dependent upon fine-grained information in the video. Instead, the dynamic image will allow the network to store a general representation of an action, ideally boosting k-shot learning accuracy.

We utilize the Memory-Augmented Neural Network model as described in Section 3.3. Our key contribution is the modification of the controller portion of the MANN. Santoro *et al*. use a flattened image as input to the controller LSTM. Instead, we use the unflattened normalized image as input to a convolutional neural network encoder that generates an encoding of the same dimensionality as the rows of the memory matrix. This network is trained jointly with the MANN. The encoder is useful for two reasons. First, because this neural network is optimized jointly with the MANN, an encoder is able to extract the most salient properties of the input image that can be stored in memory. Secondly, it drastically reduces the dimension of the input representation, which in turn, decreases the number of the parameters required for the LSTM. The full setup for the MANN network and its use for k-shot learning for action recognition can be seen in Figure 3.

Before training and evaluating the MANN, we fix two key hyperparameters: the number of unique classes in a sequence n and the length of a sequence l. To generate a sequence, we randomly sample n action classes from the train or test action set and then sample l examples from the action videos that are of the same action as one of the n chosen actions. At evaluation time, the network must perform a n-way classification task for every example in a sequence. Thus, the setup is a simplification of a general k-shot learning setup because there is a restriction on the max number of actions the network will see.

Our algorithm was implemented in Tensorflow [1]. The implementation of the MANN was adapted from an implementation by Xihan Li [16].

5. Experiments

Baseline. We consider a naive baseline for k-shot learning to compare our approach to. Given a sequence of dynamic images, we flatten each dynamic image to represent a feature vector and then perform k-means clustering on these vectors. The number of clusters is set to the hyperparameter denoting the number of unique examples per sequence. We note that this baseline is not a completely fair comparison with our approach, since the sequence is not processed sequentially and the clustering algorithm has access to the full sequence. However, it serves as a useful upper bound on the performance that a naive clustering approach can give.

Input Representations. While we experimented with representing the video as a set of frames as opposed to a dynamic image, we encountered several hardware limitations. Firstly, video representations as input had a loading time of around 15 minutes for a small batch size of 8. Secondly, we did not have enough memory on GPU to store these videos when passing through the MANN architecture. Thus, we only chose to use the dynamic image representation of the video.

Controllers. We experimented with various controllers, some of which failed due to hardware limitations. Specifically, we planned on using the state-of-the-art video action recognition network, the I3D architecture, as a controller for the MANN, but found that the 3-D convolutions and the total number of trainable parameters was too high to feasibly train with our available hardware. Thus, we focused on various implementations of the convolutional encoder discussed in Section 4. Specifically, we experimented with an encoder that follows the AlexNet architecture [14] (both pre-trained on ImageNet [5] as well as trained from scratch) or the VGG19 architecture [19]. The number of trainable parameters in the network seemed to be a key criteria to the success of the system. Encoder architectures with too many parameters, such as the VGG19 architecture and the default LSTM with no encoder performed poorly. Recall from Section 4 that introducing AlexNet decreases the overall

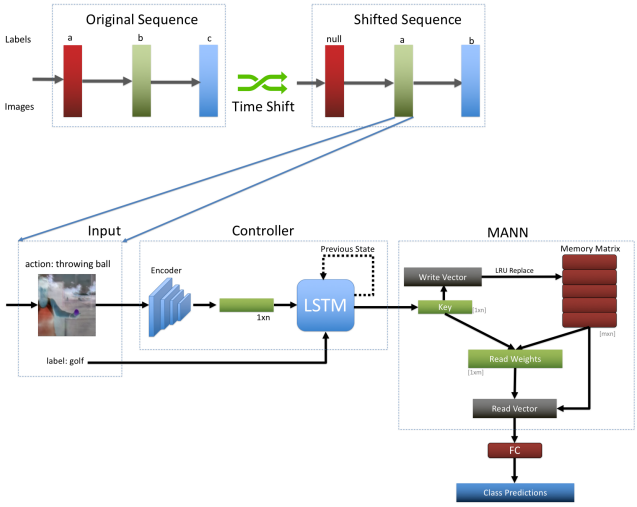


Figure 3: MANN Architecture for Action Recognition

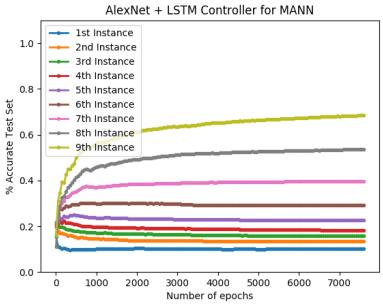


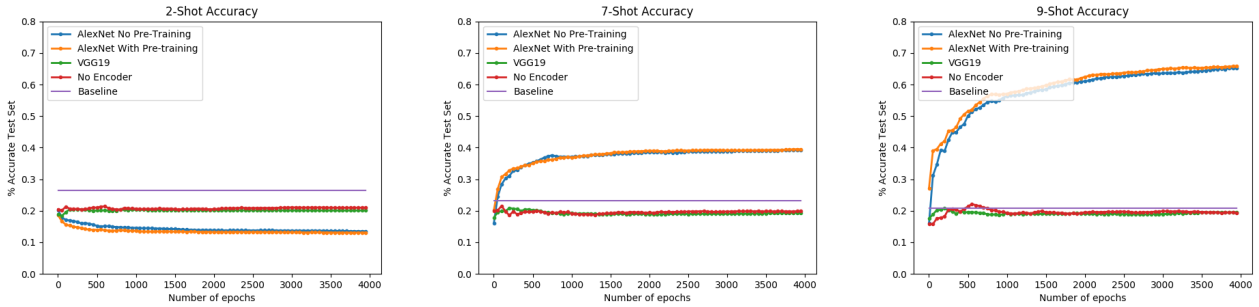
Figure 4: Performance of AlexNet with all k-shot values

Controller Architecture	*Total # of Trainable Parameters*
AlexNet + LSTM	$\approx$ 4 million
LSTM	$\approx$ 10 million
VGG19 + LSTM	$\approx$ 25 million

Table 1: Comparison of # of Trainable Parameters for MANN

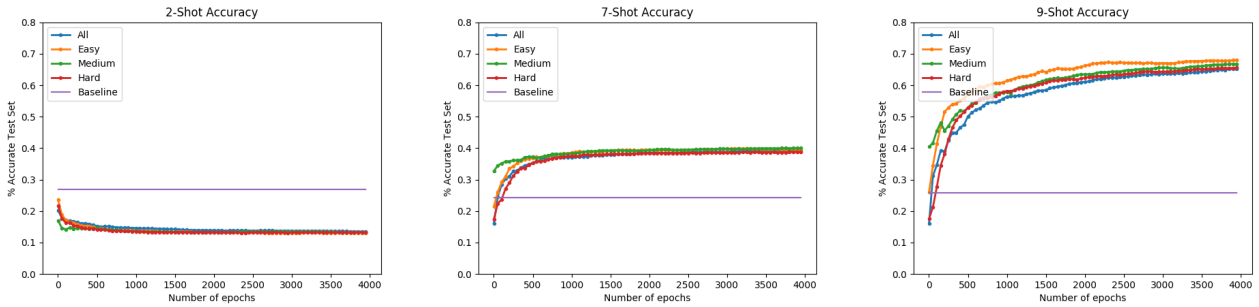
number of trainable parameters since the input passed into the LSTM is much smaller than the size of the flattened input image. The number of total parameters using these encoders is shown in Table 1. While adding more parameters gives more expressive power to the network, training is more difficult since there are more parameters to optimize. This is why we hypothesize we did not get good results with the default LSTM and VGG19 controller architectures. In the k-shot learning case, we see the differences between the controllers in Figure 5. When following the setup by Santoro *et al*. of using flattened images in conjunction with an LSTM, the results show that a pure LSTM approach is not sufficient for images with a diverse distribution like the Kinetics dataset. The Omniglot dataset [15] consists of black characters on a pure white background, which gives a much less diverse distribution of input data than the Kinetics dataset of real-world action videos. We found that our encoder module improved results not only for our action recognition task, but also on the Omniglot dataset by improving the results that were obtained by Santoro *et al*. when running for a low number of epochs. This improvement is demonstrated in Figure 7. Comparing the controllers, we see that the AlexNet controller (both pre-trained and not pre-trained) performs the best in the k-shot learning case for high k. In fact, the VGG19 and default controller perform near random accuracy for 10-shot learning. As described above, we hypothesize this is due to the large number of trainable parameters. All of the controllers have difficulty in the 1-shot case, probably because the number of epochs trained was not extremely high. However, as seen in Figure 4, k-shot accuracy steadily increases as k gets larger, which is expected.

Effect of Sequence Length and Number of Unique Classes. The sequence length and number of unique classes per sequence are intrinsic properties of the MANN setup. In Santoro *et al*., the sequence length and number of classes were set to 5 and 35 respectively for the Omniglot dataset. We hypothesized that as the number of classes increases, the k-shot recognition task becomes harder. To



(a) (b) (c)

Figure 5: Comparison of various controllers



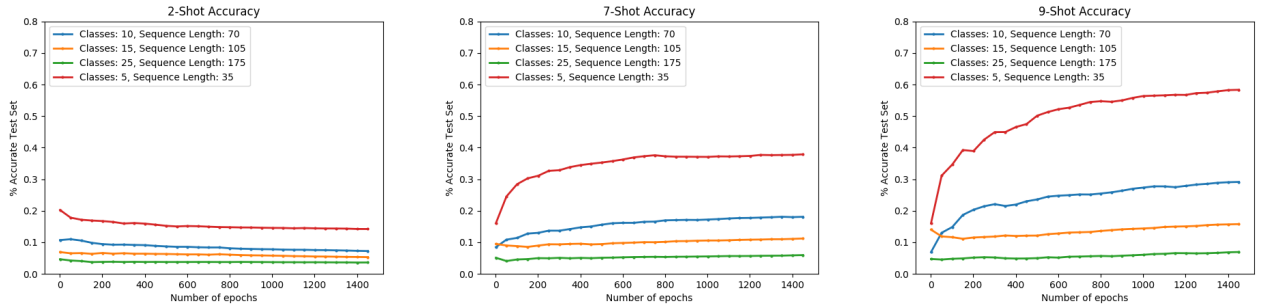
(a) (b) (c)

Figure 6: Action Difficulty Levels with controller: AlexNet trained from scratch

Figure 7: 1-shot Performance on Omniglot with added encoder

ensure that the comparisons are fair between class tests, we increase the sequence length to maintain a ratio of 7 between the sequence and the number of unique classes in a sequence. From Figure 8, we can see a large drop in performance as the number of classes is increased from even 5 to 10 classes. We believe that this decrease can be due to thrashing in memory that occurs when classes that haven't been seen recently are repeatedly evicted and placed back into the external memory, though we leave thorough verification of this guess for future work.

Dataset Segmentation. We initially hypothesized that the k-shot learning accuracies would be very biased by the specific classes chosen to be in a sequence. If these actions were very easy to identify and discriminative enough from each other in a sequence, the evaluation accuracies would be much higher than a case where actions in a sequence are very similar to each other. To dive deeper into this, we segment the dataset into 3 subsections of actions: *easy* actions, *medium* actions, and *hard* actions. To determine this classification, we sampled 10 random examples for each of the 400 actions in the Kinetics dataset and obtained classifications given by the state-of-the-art action recognition architecture, the I3D architecture [4]. The action class was classified as *easy* if the I3D network predicted the correct action label greater than 75% of the time, *medium* if accuracy in between 50% and 75%, and *hard* otherwise. After performing this split, we obtained 187 easy actions, 109 medium actions, and 104 hard actions.



(a) (b) (c)

Figure 8: Variation of MANN Number of Unique Classes and Sequence Length

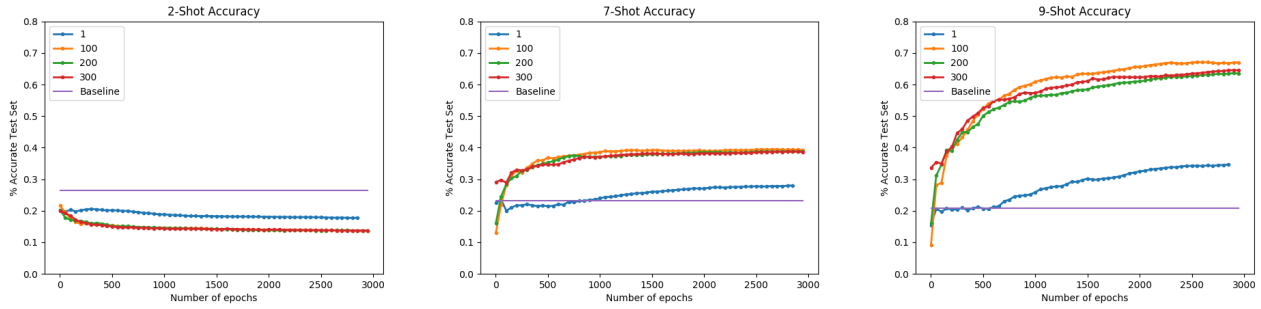
The 50-50 train split was then performed on every action difficulty level independently and we trained a MANN separately for performing k-shot classification for easy, medium, and hard actions. Figure 6 shows the performance of the AlexNet controller for the segmented datasets. The difference between the performance on the segmented datasets was not as pronounced as we expected; however, in the 10-shot case, there is still a clear hierarchy which suggests that the network performs best on easy actions and worse on medium and hard actions. Because the network is tasked with a 5-way classification task if the number of unique classes in a sequence is 5, the sequences are heavily dependent upon the actual classes sampled. Presumably, an action is classified as "hard" because the I3D architecture typically confuses this action with another very similar action label. If these two actions are not sampled together in the same sequence, the MANN might not have as much difficulty classifying this action in a 5-way classification task as the I3D architecture would have in a 400-way classification task. This suggests that in the k-shot setup, a model does not suffer as significantly as the I3D does from encountering 'difficult' actions merely due to the problem setup.

Memory Size and LSTM Hidden Neurons. An important facet of the MANN architecture is the number of hidden neurons in the LSTM, controlling the amount of 'internal' memory, and the external memory size. Figure 9 and Figure 10 show the effects of varying LSTM hidden neuron numbers and varying external memory sizes respectively. The effect of the hidden layers of the LSTM is clear, highlighting the importance of internal memory. Decreasing the number of hidden LSTM neurons to 1 hinders performance greatly, reducing 9-shot performance to near-random performance. The effect of memory size is more subtle. This is likely due to the ratio between the sequence length and the memory size. As we increase the memory size, it is unclear whether the MANN is writing to all sections of memory, especially given a sequence length much smaller than the size of memory. In our case, our sequence length is 35, so increasing the memory matrix size to 256 from 128 might not have direct gains in accuracy. However, the effect of memory size is clearly non-trivial, because if we decrease the memory size to only 2 rows, 9-shot performance is significantly worse than a larger memory size. In the case where the external memory is only 2 rows, it is likely that most of the gains in 9-shot accuracy arise from the use of internal memory in the LSTM. We make an interesting observation in the 9-shot learning case, which is that a memory matrix storing rows of size 40 performs slightly worse than a memory matrix storing rows of size 20. This is counter-intuitive, since the rows should correspond to memories, and a 'shorter' memory should be able to store less information and thus, perform worse. We were unable to come up with a convincing hypothesis for this phenomenon.

Ablation Study on Effect of Normalization. With the introduction of the convolutional encoder, we observed that normalization of the input image was crucial to achieving a good k-shot learning accuracy, as shown in Table 2. This table plots the highest test k-shot accuracy achieved during training before and after normalizing all input images. Normalizing input images has widely been shown to be critical to success of convolutional neural networks, since inputs must be of the same scale in order for the network to learn a meaningful function from input space to label space. With normalization, we see that the maximum 9-shot accuracy achieved skyrockets from 0.46 to 0.92.

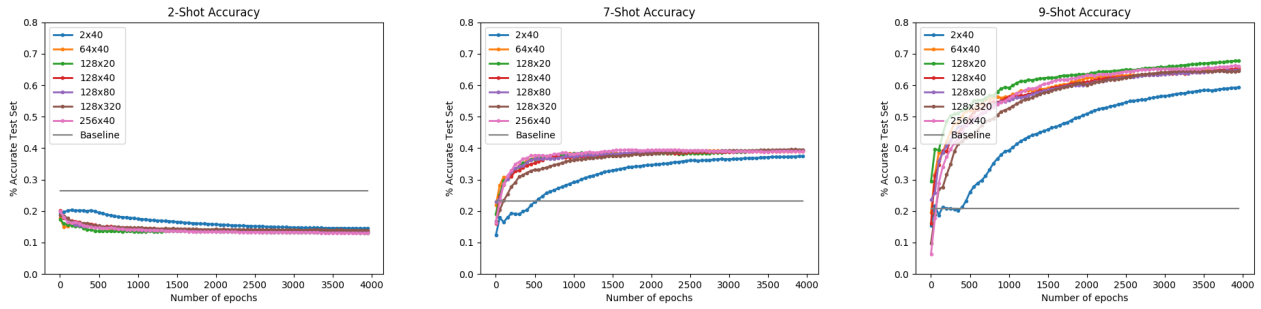
5.1. Discussion and Future Work

The k-shot recognition task is relatively unexplored in the action recognition domain. We hope that with the introduction of techniques like Memory Augmented Neural



(a) (b) (c)

Figure 9: Effect of # of LSTM Hidden Neurons



(a) (b) (c)

Figure 10: Effect of External Memory

Variant	1-*shot*	3-*shot*	5-*shot*	7-*shot*	9-*shot*
No Norm	0.25	0.23	0.28	0.35	0.46
Norm	0.20	0.24	0.32	0.57	0.92

Table 2: Ablation Study: Normalization vs. No Normalization

Networks, the possibility of low shot recognition can be fully explored. The limitations of Neural Networks requiring massive amounts of data would be greatly reduced with networks that are able to learn from few examples.

One of the natural areas of future work is to integrate a video based controller with the MANN. With the hardware memory being the limiting factor, the good performance with just the dynamic images shows the potential of extending this work to use the full representational power of videos. The video representations would provide a way to capture the temporal structure of the video that is not present in the dynamic image space. Temporal structure is especially relevant for actions where a single frame of a video cannot reliably determine an action (e.g. determining the actions of getting in and out of a car from a single frame).

The performance of the 1-shot case is drastically different to the 9-shot performance. Whilst we hypothesize that this is due to the low amounts of epochs, finding techniques that lower the training time is a critical part of making the architecture successful for multiple domains. While the encoder provides significant gains to 7-shot and above, the performance of the MANN with lower number of examples seen prior is quite poor with results being below random. The generation of the sequences is another area of further exploration, with the current methodology being random sampling of classes.

On the dataset side, it would be interesting to see how the MANN approach performs on other popular action datasets like HMDB and UCF in comparison to the Kinetics dataset. While we experimented with a 50-50 split, an area of further exploration is to compare the effect of having limited classes available for training or performing testing on fine-grained action classes to prove the robustness of the method.

6. Conclusion

We attempted to perform k-shot learning using Memory Augmented Neural Networks on the action recognition domain, specifically on the Kinetics dataset. We introduce using an encoder to extract the salient features from dynamic image representations of the videos. Our experiments show considerable gains in accuracy from using memory for k-shot action recognition, reaching up to 78% accuracy in the 9-shot learning case.

References

[1] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, et al. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *arXiv preprint arXiv:1603.04467*, 2016. 5

[2] H. Bilen, B. Fernando, E. Gavves, A. Vedaldi, and S. Gould. Dynamic image networks for action recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3034–3042, 2016. 1, 2, 5

[3] S. Caelles, K.-K. Maninis, J. Pont-Tuset, L. Leal-Taixé, D. Cremers, and L. Van Gool. One-shot video object segmentation. In *CVPR 2017*. IEEE, 2017. 2

[4] J. Carreira and A. Zisserman. Quo vadis, action recognition? a new model and the kinetics dataset. *arXiv preprint arXiv:1705.07750*, 2017. 1, 7

[5] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 248–255. IEEE, 2009. 5

[6] J. Donahue, L. Anne Hendricks, S. Guadarrama, M. Rohrbach, S. Venugopalan, K. Saenko, and T. Darrell. Long-term recurrent convolutional networks for visual recognition and description. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2625–2634, 2015. 2

[7] C. Feichtenhofer, A. Pinz, and A. Zisserman. Convolutional two-stream network fusion for video action recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1933–1941, 2016. 1, 2

[8] A. Graves, G. Wayne, and I. Danihelka. Neural turing machines. *arXiv preprint arXiv:1410.5401*, 2014. 3

[9] Y. Guo, W. Ma, L. Duan, Q. En, and J. Chen. Human action recognition based on discriminative supervoxels. *2016 International Joint Conference on Neural Networks (IJCNN)*, pages 3863–3869, 2016. 2

[10] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997. 3

[11] S. Ji, W. Xu, M. Yang, and K. Yu. 3d convolutional neural networks for human action recognition. *IEEE transactions on pattern analysis and machine intelligence*, 35(1):221–231, 2013. 1, 2

[12] W. Kay, J. Carreira, K. Simonyan, B. Zhang, C. Hillier, S. Vijayanarasimhan, F. Viola, T. Green, T. Back, P. Natsev, et al. The kinetics human action video dataset. *arXiv preprint arXiv:1705.06950*, 2017. 5

[13] G. Koch. Siamese neural networks for one-shot image recognition. 2015. 2

[14] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012. 5

[15] B. M. Lake, R. Salakhutdinov, and J. B. Tenenbaum. Human-level concept learning through probabilistic program induction. *Science*, 350(6266):1332–1338, 2015. 2, 3, 6

[16] X. Li. Memory-augmented neural networks tensorflow implementation. Github: https://github.com/snowkylin/ntm. 5

[17] V. Mnih, N. Heess, A. Graves, et al. Recurrent models of visual attention. In *Advances in neural information processing systems*, pages 2204–2212, 2014. 3

[18] A. Santoro, S. Bartunov, M. Botvinick, D. Wierstra, and T. Lillicrap. One-shot learning with memory-augmented neural networks. *arXiv preprint arXiv:1605.06065*, 2016. 1, 2, 3

[19] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014. 5

[20] A. J. Smola and B. Schölkopf. A tutorial on support vector regression. *Statistics and computing*, 14(3):199–222, 2004. 2

[21] O. Vinyals, C. Blundell, T. Lillicrap, D. Wierstra, et al. Matching networks for one shot learning. In *Advances in Neural Information Processing Systems*, pages 3630–3638, 2016. 2

[22] H. Wang and C. Schmid. Action recognition with improved trajectories. In *Proceedings of the IEEE international conference on computer vision*, pages 3551–3558, 2013. 1, 2

[23] J. Yue-Hei Ng, M. Hausknecht, S. Vijayanarasimhan, O. Vinyals, R. Monga, and G. Toderici. Beyond short snippets: Deep networks for video classification. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4694–4702, 2015. 2