COS 429: Class Project Report
Examining Motion With a Neural Net

Ashley Kling, Ashley Thomas, Mark Martinez, Natalie Wilkinson

January 17, 2017

1 Introduction

Deep learning has revolutionized machine learning and recently wrought a similar change in computer vision. Feature vectors for identifying parts of an image used to be painstakingly handcrafted, but now can be generated by simply supplying a convolutional neural network trained with enough data. Additionally, because the data is being used directly to generate these feature vectors, the network can find important features that may not have been apparent to a person determining them by hand.

For our project, we wanted to take on the challenge of using deep learning. However, instead of identifying objects in single images as we saw in class and in the homeworks, we wanted to classify motions across multiple frames of a video. We decided, based on the current emphasis placed upon deep learning, to use a neural network to implement this classification. Specifically we decided to use a convolutional neural network to identify an action that a person is undertaking in any given video. The types of actions we planned to classify are simple individual motions such as a cartwheel, a handstand, or jumping.

2 Overview

There are a number of ways in which to address the problem of classifying motions. The paper by Karpathy [1] highlights a number of ways to attempt the problem of classifying motion with a convolutional neural network. They start by treating the videos as a sequence of short, fixed-size images, or frames. Thus the problem becomes finding a way to incorporate the temporal information alongside the spatial information present in the images. The baseline they use ignores the temporal information completely, running the neural network on a single frame of the video. They call this architecture Single Frame.

The other three architectures they investigate incorporate the temporal information in various ways. In Early Fusion, they extend the input to be of size $w \times h \times rgb \times T$, where T is the temporal information. Thus instead of providing the neural network with a single rgb image, the neural network is provided with T rgb images. In Late Fusion, Single Frame networks are placed alongside each other in the network and merged at the first fully connected layer. In Figure 1, the late Fusion diagram has 2 such Single Frame networks, spaced 15 frames apart. Thus while neither individual frame has knowledge of any temporal information, the entire network can detect motion by comparing the difference between the single-frame networks. The final architecture they

discuss, Slow Fusion, is a combination of Late and Early Fusion, where the additional dimension of time is included in the input, as in Early Fusion, and a number of these Early Fusion networks are combined as in Late Fusion. We decided that the best route, given the networks we have access to, was to use the Late Fusion architecture.

Figure 1: CNN Architectures

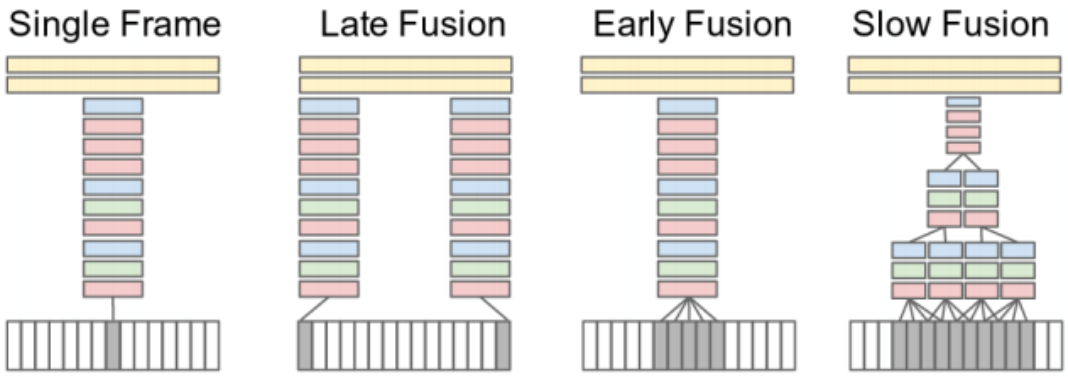


Image originally from the Karpathy paper [1]

3 Methods

For training and testing data, we used HMDB: a human motion database from the Serre Lab [2], which contains 7,000 videos of 51 different actions. The videos come from a variety of sources, including commercial movies and amateur sources such as YouTube. We decided to classify videos using HMDB videos of the following four actions: cartwheel, punch, climb, and jump. We picked these actions because they motion can be enacted simply by moving body parts and do not require small objects or another interacting human. Figure 2 shows an example of a series of frames of the action jump.

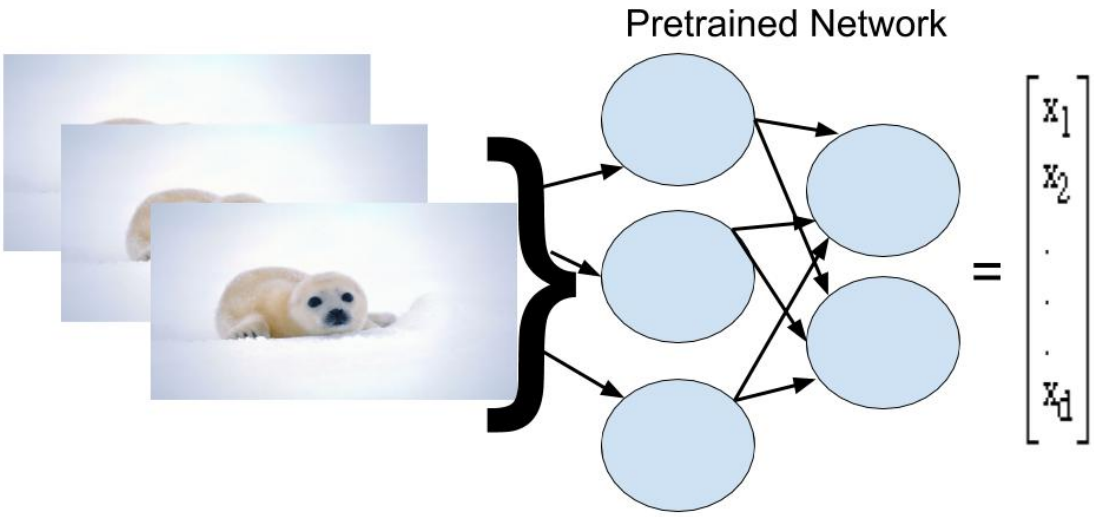
Figure 2: 3 consecutive frames of jumping action

(a) Jump 1 **(b)** Jump 2 **(c)** Jump 3

In order to classify the videos, we first had to extract the images from the videos. To do so, we separated the videos into sets of 15 continuous frames. For each of these sets, we took 3 evenly spaced frames to feed to our classifier. As an example, images 5, 10, and 15 would make up a typical set where we extracted frames 1 to 15. We split up the videos in this way in order to have more data for each action, and the spacing was needed for better indication of motion. We ensured that although we made it possible for segments of the same video to be used in either training or test, no segment was used for both actions, so all of our sets are unique.

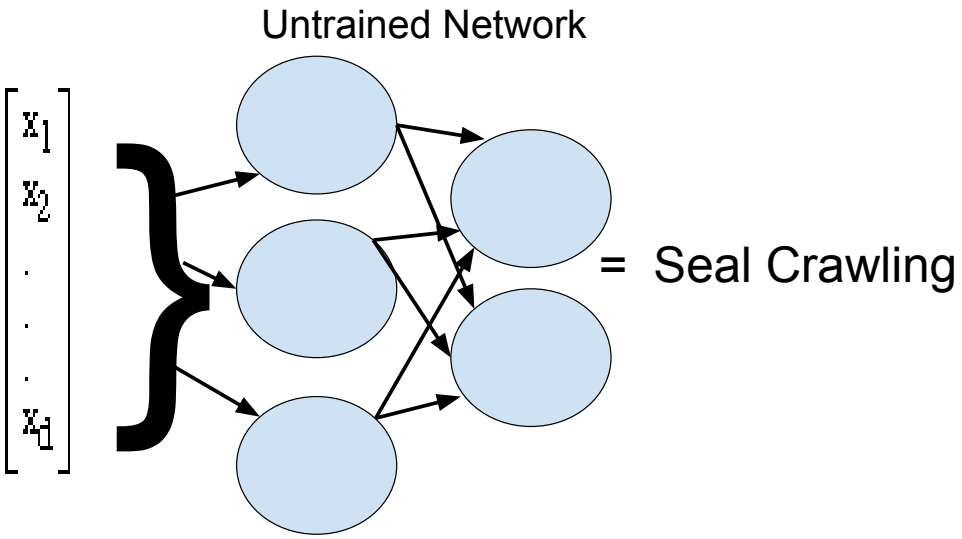
As we did not have sufficiently large enough data to fully train a CNN from scratch, we used a pre-trained CNN called VGG and adapted it to our purposes instead. We chose VGG because it was simple to modify for our purposes and it has a classification for people. Based on [1], we decided that Late Fusion would fit our methods best. We implemented this by taking a VGG network and deleting the fully-connected layers at the end. We then extracted each set of three frames and sent each frame in the set through a modified VGG network. This step is illustrated in Figure 3. We then concatenated the outputs of the three networks and ran them through our own classifier to produce the final result. This step is illustrated in Figure 4. Both the VGG network, and the network we created were done using the matconvnet tools [4] introduced in the homeworks.

Figure 3: Pretrained Network



Three images of an action from one video are each spaced out 5 frames apart and inputted into a pretrained network. This step is used to detect the objects in the images. The output for each image is concatenated together to serve as inputs to our new network.

Figure 4: Pretrained Network



Using the output from fig. X we then use it to train and test the new layers of the network to classify motion.

4 Results

In the attempt of creating a convolutional neural network which could classify the videos with decent accuracy, we created two different CNNs and then applied different learning rates to each. Our baseline neural network is shown in Figure 5a, while our modified neural network is show in Figure 5b. The main difference between the two neural networks is the rate at which our data size decreases down to 1, which changes the number of parameters drastically. The baseline network has approximately 140,000 parameters, while our modified network has 330,000. The first row in each figure indicates the layers of the network, with the baseline network running through more relu and convolutional layers than the modified network. Our resulting data depth for each network is 4, as we are producing 4 different classifications for each of the 4 actions.

To produce our results, we ran each of our networks with two different learning rates for 2000 epochs, producing the results shown in Figure 6. Each diagram has an objective graph and a top1err graph. The objective graph is simply the output of the loss function. The lower the loss, the better our network does. The top1err is the percentage that our network was incorrect in determining the classification. Figure 6a and Figure 6c contains the results from our baseline network shown in Figure 5a, when run with a learning rate of 0.001 and 0.01 respectively. Figure 6b and Figure 6d contains the results for our modified network shown in Figure 5b, with a learning rate 0.001 and 0.01 respectively.

Figure 5: Network Diagram

(a) Baseline Network

type	input	conv	relu	conv	relu	conv	relu	conv	relu	conv	mpool	relu	softmxl
name	n/a	layer1	layer2	layer3	layer4	layer5	layer6	layer7	layer8	layer9	layer10	layer11	layer12
support	n/a	1	1	1	1	3	1	1	1	3	3	1	1
filt dim	n/a	768	n/a	100	n/a	100	n/a	50	n/a	50	n/a	n/a	n/a
filt dilat	n/a	1	n/a	1	n/a	1	n/a	1	n/a	1	n/a	n/a	n/a
num filts	n/a	100	n/a	100	n/a	50	n/a	50	n/a	4	n/a	n/a	n/a
stride	n/a	1	1	1	1	1	1	1	1	1	3	1	1
pad	n/a	0	0	0	0	0	0	0	0	0	0	0	0
rf size	n/a	1	1	1	1	3	3	3	3	5	7	7	7
rf offset	n/a	1	1	1	1	2	2	2	2	3	4	4	4
rf stride	n/a	1	1	1	1	1	1	1	1	1	3	3	3
data size	7	7	7	7	7	5	5	5	5	3	1	1	1
data depth	768	100	100	100	100	50	50	50	50	4	4	4	1
data num	10	10	10	10	10	10	10	10	10	10	10	10	1
data mem	1MB	191KB	191KB	191KB	191KB	49KB	49KB	49KB	49KB	1KB	160B	160B	4B
param mem	n/a	300KB	0B	39KB	0B	176KB	0B	10KB	0B	7KB	0B	0B	0B

parameter memory|533KB (1.4e+05 parameters)|
data memory| 2MB (for batch size 10)|

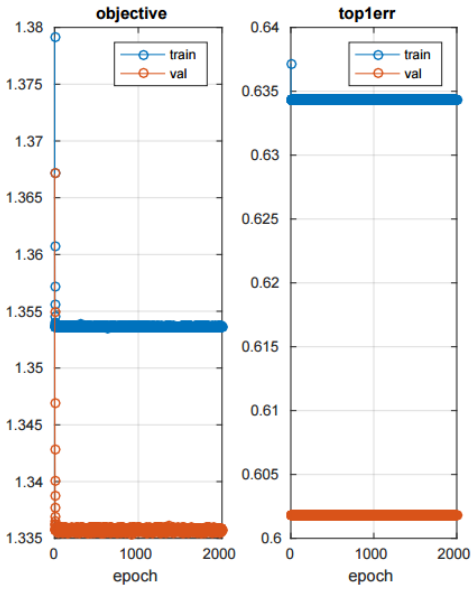
(b) Modified Network

type	input	conv	mpool	relu	conv	mpool	relu	conv	relu	softmxl
name	n/a	layer1	layer2	layer3	layer4	layer5	layer6	layer7	layer8	layer9
support	n/a	2	2	1	2	2	1	1	1	1
filt dim	n/a	768	n/a	n/a	100	n/a	n/a	50	n/a	n/a
filt dilat	n/a	1	n/a	n/a	1	n/a	n/a	1	n/a	n/a
num filts	n/a	100	n/a	n/a	50	n/a	n/a	4	n/a	n/a
stride	n/a	1	2	1	1	2	1	1	1	1
pad	n/a	0	0	0	0	0	0	0	0	0
rf size	n/a	2	3	3	5	7	7	7	7	7
rf offset	n/a	1.5	2	2	3	4	4	4	4	4
rf stride	n/a	1	2	2	2	4	4	4	4	4
data size	7	6	3	3	2	1	1	1	1	1
data depth	768	100	100	100	50	50	50	4	4	1
data num	10	10	10	10	10	10	10	10	10	1
data mem	1MB	141KB	35KB	35KB	8KB	2KB	2KB	160B	160B	4B
param mem	n/a	1MB	0B	0B	78KB	0B	0B	816B	0B	0B

parameter memory|1MB (3.3e+05 parameters)|
data memory| 2MB (for batch size 10)|

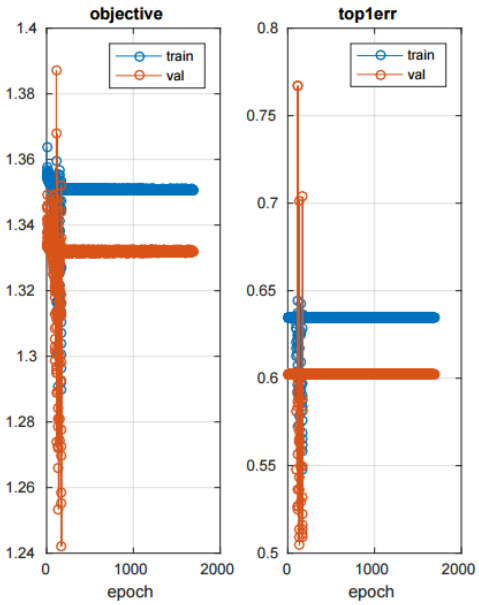
Figure 6: Network Results

(a) Baseline Network



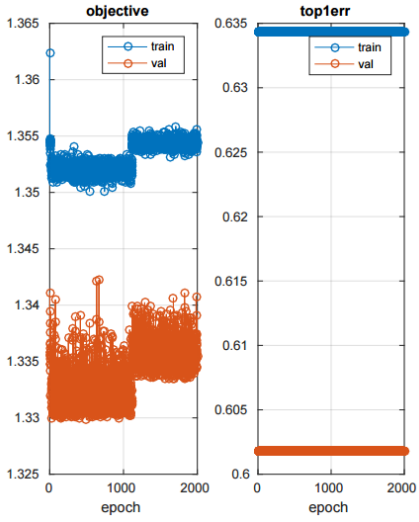
Baseline Network with learning rate of 0.001

(b) Modified Network



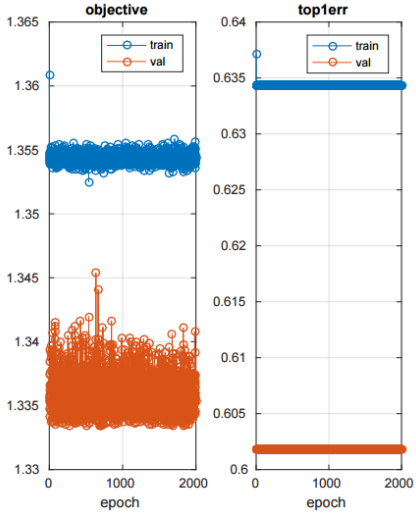
Modified Network with learning rate of 0.001

(c) Baseline Network



Baseline Network with learning rate of 0.01

(d) Modified Network



Modified Network with learning rate of 0.01

5 Analysis

In Figure 6a we utilized a network that has the traditional odd numbered values for convolutions, and only one pooling layer at the very end of the network before the softmax layer. Because our data size was fairly small to begin with we used extremely small convolutions, with a filter dilation size of 1×1. We see that the network started learned in the earlier epochs, but failed to learn further, possibly due to lack of data to train on. The network plateaued after about 20 epochs and maintained the same level of classification for both the test and the training set.

The modified network from Figure 6b used parameters in the layers that were non-traditional. For instance, we included two convolutional layers with an even filter dilation size, of size 2×2. However, it seems that this network performed slightly better than the baseline network with a learning rate of 0.001, especially looking at the first few hundred epochs, as it dropped down to a loss of 1.24 in Figure 6b. Unlike the other networks, this network's error decreased for both the training and test sets in the first few hundred epochs, meaning that the network was actually learning and improving. Unfortunately, after these first few hundred epochs, the error plateaued around 1.335, similar to the baseline network. It's possible that because the convolutions' filter dilation size is even we are mostly utilizing the tails of the function to convolve on the data, but because the tails of the function drop off to zero quickly it's possible that unless something is particularly prominent, like a change in motion, it would not remain across the images.

In Figure 6c we use the same baseline network, and increase the learning rate to 10 times the original. This network seems to oscillate more than when run with a lower learning rate, with the loss function having a much greater range when it plateaus than the network with the lower learning rate. This is likely due to the network overshooting, and being unable to find the best parameters. The jump in the objective is unusual and may be caused by overfitting the data once a sufficient number of epochs has passed. In Figure 6d we use the modified network and a larger learning rate of 0.01. This network stabilizes faster than the network with the smaller learning rate. However, the larger learning rate almost certainly contributed to the variance in the objective by making it harder to reach local minima. Both modified networks appear to converge to the same value of around 1.355 and 1.335 for the train and val respectively.

Because we built our network specialized to detect motion it has a number of strengths compared to a more generic CNN. We use three images spaced out over 15 frames to classify one motion so our classifier has the ability to detect motion by analyzing the differences in the main image whereas a single-frame CNN would not have this temporal information. Our CNN is stronger than a CNN that was trained on only the motion data because we utilized the VGG network to first detect the objects in the images and took the outputs from before the last fully connected layer as inputs to our network. Because of this we were able to use our chimeric network to find the high level information of motion instead of detecting objects. Had we only used a non-chimeric network we would not have enough data to detect motion and the objects in the images. In the network with the best set of parameters our network did well given how few data we had for each motion. Had we had more data our network would have likely performed better, with values falling below 60% error. Unfortunately, due to time restrictions, finding and preprocessing a sufficient amount of data to produce a fully trained neural network was not feasible. VGG was a good network to work with given the scope of the project because we had used it in the homework assignment, but it had limitations when detecting specific parts of the human body. Had we used a pre-trained network that isolated body parts such as an arm of a person, as opposed to only being able to classify the person, we would have likely had better results in classifying the motion.

6 Future Work

There are plenty of ways to extend and improve upon this work that wed like to explore upon in the future. First, although we used Late Fusion to train our CNN, the Karpathy paper pointed out that they obtained better results from using Slow Fusion. Exploring the differences between using Late Fusion, Slow Fusion, and Early Fusion in our algorithm could be informative. Also, it could be useful to experiment with the number of frames that we run through the pre-trained CNNs. Currently we use three, but we may get better results when examining more at a time. Another factor that could be affecting our results is the fact that because a lot of our data came from YouTube and this isnt particularly consistent, some actions were performed faster in some videos than others, which may affect the classification. Currently our stride between frames is five, but experimenting with varying number of frames could correct for any error induced by inconsistencies related to the speed of the action. Also, of the pre-trained networks available, we picked VGG because it seemed to have the greatest number of desirable features for our project. However, it would be interesting to see if using other networks instead of using VGG causes any substantial differences in outputs. Finally, to improve on the size of the test data, we could also crop and reverse the video so that we have more test data to input into the neural network, as our data size is currently $\tilde{2}000$, and this is a fairly small size to work with.

7 Conclusions

In experimenting with making a neural net to detect human motion, we learned quite a few things. First, we found that although training for more epochs is often a good idea, sometimes it's possible to train for too many epochs. We also learned that a lower learning rate improved the results by making it easier for the network to find local minima. Having too large a learning rate leads to too much variance. Furthermore, our nine-layered network did better than our twelve-layered network, potentially due to the change in our convolutions' filter dilation, or to the rate at which we decreased our data size which led to an increase in parameters. Futhermore, because our network was specially crafted to include a temporal element, it was able to detect motion patterns better than a regular network. In the future we could improve on this project by using a more specialized pre-trained network, better data, and Slow Fusion instead of Late Fusion.

References

[1] Andrej Karpathy, George Toderici, Sanketh Shetty, Thomas Leung, Rahul Sukthankar, and Li Fei-Fei. Large-scale video classification with convolutional neural networks. In *CVPR*, 2014.

[2] H. Kuehne, H. Jhuang, E. Garrote, T. Poggio, and T. Serre. HMDB: a large video database for human motion recognition. In *Proceedings of the International Conference on Computer Vision (ICCV)*, 2011.

[3] Evan Shelhamer, Jonathan Long, and Trevor Darrell. Fully convolutional networks for semantic segmentation. *CoRR*, abs/1605.06211, 2016.

[4] A. Vedaldi and K. Lenc. Matconvnet – convolutional neural networks for matlab. In *Proceeding of the ACM Int. Conf. on Multimedia*, 2015.