Kendall Weihe

CS463G Fall 2016 final write-up

**Introduction**

For my final project I decided to recreate parts of a deep neural network developed in **[CITE]** Karpathy et al. [1]. Over the last year I have advanced my software engineering skills in the field of deep neural networks. My final project in the *University of Kentucky’s* *CS 463G Artificial Intelligence* is an opportunity to showcase the skills I have learned by programming the networks outlined in Karpathy et al. [1].

I have recreated papers before, but this project is the first time I have recreated a paper and used the same dataset. By using the same dataset, I am able to compare my results directly to the results in the paper. The dataset used in **[CITE]** and this paper is the UCF-101 Action Recognition Dataset **[cite?]**. I ran into several problems from the UCF-101 dataset, discussed in the Data section.

**[CITE]** Karpathy et al. [1] published their work in the *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* in 2014. Karpathy et al. [1] proposed a set of Convolutional Neural Networks that took advantage of the spatial-temporal features of the videos in the UCF-101 dataset. Their models include, what they refer to as *Time Information Fusion in CNNs*. The networks I developed use the same Time Information Fusion architecture. However, the networks I developed are not identical to those in **[CITE]** Karpathy et al. [1] – discussed further in the Model section.

**Related Work**

**[CITE]** Karpathy et al. [1] developed three different Time Information Fusion networks to take advantage of spatial-temporal features of videos: late fusion, early fusion, slow fusion. Figure 1 shows a graphical representation of how the networks look. The deep neural networks are made up of convolutional layers, max pooling layers, fully connected layers, and a softmax classifier. A fourth network – the single-frame network – was developed as a baseline architecture to measure the contribution of the time information fusion networks. The fourth network takes a single video frame as an input image – not giving the network any temporal information.

The late fusion network is composed of two single-frame networks with the addition of two final fully connected layer’s. The first frame of the video is passed through a single-frame network and the last frame of the video is passed through a second single-frame network. The output of each single-frame *network* is merged and then passed through two fully connected layers.

The early fusion network is the same as the single-frame network, except the input is a volume of frames. Karpathy et al. [1] developed the network to accept a volume of frames by modifying the first convolutional layer’s filters to be 11x11x3xT where T is some temporal extent (T = 10).

The slow fusion network has the same volume of frame inputs as early fusion (T = 10), but slowly fuses chunks of frames together across discrete convolutions. The first layer of the network is composed of four separate convolutions, where each convolution inputs T = 4 frames. For example, convolution #1 input volume is made up of frames [0-3], and convolution #2 input volume is made up of frames [2-5]. The second layer fuses the outputs of the first layer together with T = 2. For layer 2, there are two separate convolutions. For example, layer two convolution #1 fuses the outputs from layer one convolution #1 and convolution #2 together. The third layer then fuses together the two outputs from layer two. The third layer therefore has access to information across all 10 frames. The idea behind the slow fusion network is to slowly fuse temporal information together such that higher layers get access to progressively more global information **[CITE]** [1].

**Data**

The UCF-101 Action Recognition Dataset is composed of 101 different classes of videos. The dataset is roughly 7GB, there is a total of **[NUMBER OF VIDEOS]** videos, and each video is of resolution 320x240. My desktop work station does not have enough RAM to handle a deep neural network where the input size could be of size 320x240x10 (for early and slow fusion). Therefore, I had to do some preprocessing on the data to make it more manageable (for information on how the programs function, see README.txt). After preprocessing, I was able to manage 12 video classes – a total of 1641 videos – where each video was of resolution 80x60 (1/4 of the original resolution). Training on only 12 of the 101 classes, and downsampling the video resolution should be taken into account when comparing results of my network to **[CITE]** Karpathy et al. [1].

**Model**

I developed four networks: single-frame, late fusion, early fusion, slow fusion. The networks I developed mimic the general architecture of **[CITE]** Karpathy et al. [1], but have different hyperparameters. I developed the networks using Google’s Tensorflow deep learning API. All four networks are convolutional neural networks. Each convolutional layer is a 3D convolutional layer. The number of filters at each layer is: 4, 8, 16, 32, 64. The final layer of each network is a fully connected layer. Dropout is applied after the final fully connected layer. The networks has a batch training size of 30, a learning rate of 0.001, and a dropout probability of 0.75. All weights are initialized using Xavier initializations, and biases are all initialized to zero. All networks train for 500 epochs. The input data is randomly shuffled, and the same prediction set is used across all four networks.

**Single-Frame**

The single-frame network is composed of five sequential convolutional layers. Each convolutional layer uses a non-unit stride of two, thus halving the input dimensions after each layer (this is opposed to the max pooling layers used in **[CITE]** Karpathy et al. [1]). Batch normalization occurs after each convolutional layer. The final layer is a fully connected layer.

**Late Fusion**

The late fusion network is composed of two parallel single-frame networks, minus the final fully connected layers. The network splits the input volume into two video frames, and passes each frame into its own single-frame network. The output from each single-frame network (from the fifth convolutional layers) are concatenated along the x-axis and then passed through a fully connected layer. The fully connected layer is then passed through a dropout layer.

**Early Fusion**

The early fusion network is the same as the single-frame network, except it inputs a volume of 10 video frames as opposed to a single video frame. The Tensorflow tf.nn.conv3d() function allows me to pass a volume of frames through a three-dimensional convolution.

**Slow Fusion**

The slow fusion network is the most involved. The slow fusion network inputs a volume of 10 video frames and then splits them into 4 chunks (as described in the Related Work section). Each chunk is passed through two convolutional layers – which includes batch normalization at each layer. In other words, there are four parallel streams of convolutional layers. After the first two layers, chunks 1&2 are concatenated and chunks 3&4 are concatenated (both along the x-axis). Each of these two chunks are then passed through two more convolutional layers – layers three and four. After layer four, the two chunks are concatenated along the x-axis and passed through one final convolutional layer. The output of the fifth convolutional layer is then batch-normalized and passed through the final fully connected layer.

**Results**