End-to-End Deep Networks for Hand Gesture Recognition

CS 482/682 Final Project Report

Tanay Agarwal, Jaewon Chung, Manyu Sharma

May 5, 2018

Abstract: Gestures are an important medium for human communication. Automated recognition of human hand gestures would allow for a host of downstream applications. In the following report, we study the performance of two different deep architectures classifying the 20BN-JESTER dataset, containing 150,000 densely labeled video clips of 27 classes of hand gestures. Our implementation of a 3DCNN (C3D) reached a peak validation accuracy of 84.2 percent. Our LRCN reached a peak validation accuracy of 74.1 percent. We feel that with additional computing resources, we could have achieved better performance, but are confident that the models we developed can be used for live classification.

1 Background & Problem Statement

Hand gestures have always been an important component of human communication. We make signals with our hands to express different actions that are easily understood by anyone who sees them. The efficiency of sharing intent through hand motions has made hand gesture classification an exciting opportunity for technological advancement. Effective hand gesture recognition could have applications in several domains, such as natural user interfaces, application control, robotics, gaming, etc.

There have been some attempts to allow for hand gesture control of applications, such as the Microsoft Kinect. However, systems like this have several drawbacks since they require the purchase of dedicated hardware. They also often use pipeline methods in combination with domain-specific classification methods.

The advent of deep learning has given rise to complete end-to-end systems for common computer vision and sequence modeling problems. For example, convolutional architectures provide state-of-the-art solutions for image recognition tasks, while recurrent architectures can successfully model time-dependent features. These end-to-end systems are highly efficient at performing several steps of the traditional pipeline methods, such as feature extraction, modeling, and prediction. In this project, we aim to use deep networks to create an end-to-end hand gesture recognition system.

Convolutional neural networks have been used for image classification tasks for many years, and have been shown to achieve incredible results on datasets such as ImageNet. For example, ResNet 152 achieved 3.58% misclassification rate classifying 100,000 images into 1,000 classes [1]. One of the benefits of convolutional neural networks is translational invariance, which is due to inherent properties of convolutional and max-pooling layers present in many convolutional neural networks [2]. Thus, a natural extension of convolutional neural networks for image classifications is to use them in video classification problems.

Our goal is to maximize the validation accuracy (and potentially submit test predictions to the official leaderboards) on the 20BN-JESTER dataset collected by the company TwentyBN. The dataset consists of nearly 150,000 densely labeled videos of hand gestures, split into training, validation, and testing sets. The videos are of humans performing 27 different classes of actions, such as swiping left and zooming in with two fingers.

2 Methods & Models

We explored the effectiveness of deep networks for classifying hand gesture videos. This task presents a greater challenge than the traditional computer vision problems. Since our inputs are videos, our data has much higher dimensionality. Furthermore, not only do our models have to extract spatial features, but they also have to extract temporal features in order to classify the hand gestures effectively. These requirements naturally gave way to the use of convolutional and recurrent architectures.

The first model we implemented was a 3-dimensional convolutional neural network (3D-CNN). Unlike a 2D-CNN, such as ResNet, 3D-CNN apply 3-dimensional convolutional filters where the input data has three dimensions: x and y in the spatial dimensions and a third time dimension. The 3D convolutional kernels can effectively extract spatiotemporal features [3]. Tran et al. proposes a very simple CNN network called C3D network as shown in Figure 1. The major difference between C3D and other image classification CNNs is that the kernels are 3D. Our initial 3D-CNN has a very similar architecture to that of C3D, but we "downsized" the network due to hardware memory restrictions by removing few of the larger convolutional layers. While the C3D model does not specify any regularizations, we implemented a batch normalization layer after each existing convolution layer due to its effectiveness in preventing overfitting. Tran et al. also state temporal depth of the kernel can significantly impact predictive performance, and found an optimal depth of 3 in the UCF101 dataset. As such, we also investigated different kernel depth to find the optimal depth for the Jester dataset.

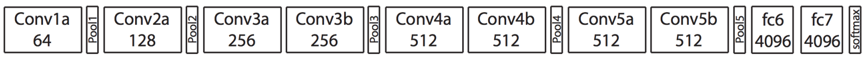

Figure 1: C3D Architecture

The second model we implemented was a hybrid neural network that combines CNN and long-term short memory (LSTM) network. These models are motivated by the fact that RNNs, such as LSTMs, are useful for understanding sequential data, such as sentences and videos. Thus, combining 3D-CNNs and LSTMs address the main downside of 3D-CNNs, which is that it has a fixed spatiotemporal receptive fields defined by the kernel size [5]. In practice, the features generated by 3D-CNNs, which can be thought of as lower dimensional representation of a video segment, can be fed into LSTM layers, which allows for greater time dependencies.

Our CNN-LSTM model is based on the long-term recurrent convolutional network (LRCN) proposed by [4]. A general overview of the model is shown in Figure 2. LRCN will take a visual input, such as a frame of a video, which is transformed to a fixed feature vector through a CNN. The feature vector is then passed to a sequence model, which consists of LSTM layers. With our video inputs, the LRCN will perform a 2D convolution on each frame, and then the features from each frame will be considered separate timestep-inputs in the context of the LSTM layer. One benefit of LRCN is that it can take sequential inputs and produce either sequential outputs (description of the inputs) or fixed outputs (visual activity recognition/classification) [4]. However, we will be mainly focused on the fixed outputs as our goal is to classify specific hand gestures.

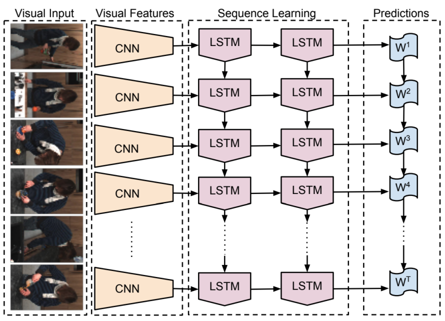

Figure 2: Long-term Recurrent Convolutional Network (LRCN) Architecture

Our CNN-LSTM model is different from the original LRCN model proposed by [4]. The original LRCN model has two "phases": the convolution network, followed by the recurrent layer. The convolution network is based on AlexNet, but we decided to implement our LRCN architecture using the convolutional model we developed for our first model (the one based on C3D). We did this because we felt it would make our comparisons more reliable, because we would be able to control and observe the effects of adding the recurrent layer to our convolutional structure. Another difference in our LRCN implementation is in our usage of the LSTM layer. The original model uses the outputs from each timestep of the LSTM layer and averages the predictions from each of these timesteps. We implemented our network to use only the final hidden vector output of the LSTM layer when producing the final probability distribution over the classes. We made this change because this is typically how recurrent layers are used, and because we want to leverage the benefits of the layer having seen all the timesteps. Using the outputs from all timesteps will negatively affect the predictions because the first several timesteps have no valuable information.

In order to use the dataset with our models, we had to do a fair amount of preprocessing. The videos all have the same 100px height, but they have varying widths and duration. We have formatted all videos to have the same width by padding those with smaller widths with empty pixels (black). For the variability in duration, we pad the video sequence with the last frame of their sequence if they are too short, and randomly sample frames if the sequence is too long. This formatting allows us to use a simpler architecture for training across all samples.

3 Experiments & Results

We ran several experiments using both our C3D and LRCN architectures in order to explore the hyperparameter space of these models. We ran all experiments on dual Tesla K80 GPUs using PyTorch's multi-GPU support. We used the entire Jester dataset of 100,000+ training examples and 14,000+ validation examples. We trained all our models for 10 epochs and recorded the highest achieved validation accuracy during this time. We measured accuracy using the Precision@1 metric, which means that a prediction is considered correct if the true class is assigned the highest probability by the model. Below we present the results and our rationales for the exploration.

For our first model (3D-CNN), we explore a variety of values for settings such as kernel height/width, kernel depth, and optimizers. We experimented with different values and combinations of kernel height/width and depth because these play a big role in the functioning of the convolutional network. Different datasets have different optimal kernel heights/widths, and so we tried different values to find the best one for the Jester dataset. The kernel depth determines the size of the spatiotemporal field from which the CNN is extracting features at each layer, and so this hyperparameters plays a big role in the kinds of time-related information that the network is able to aggregate and exploit when making classifications. We tried different values for these and collected the converged validation accuracy. We use a fixed stride of 1 for all dimensions in order to limit our hyperparameter search due to resource limitations.

C3D	**Height/width 2**	**Height/width 3**	**Height/width 4**
Depth 2	80.5	79.2	79.1
Depth 3	82.3	82.1	81.6
Depth 4	83.5	83.2	83.0

Table 1: Effect of varying kernel sizes in depth and height/width for C3D model using SGD

We also explored the performance of different optimizers for our convolutional model. The optimizer plays a crucial role in the exploration of the parameter space, and so it has a direct impact on the validation accuracy that we achieve. We used our "default" convolutional network (kernel height, width, and depth 3) to control and isolate the effect of the optimizer. We used the default recommended learning rates for each optimizer (see Table 3 below).

For our second model, we explore a variety of values for settings such as the number of layers, dropout rate, and optimizer. We experimented with different numbers of layers because this affects the capacity of the recurrent architecture. We tried different combinations of this with various dropout rates to see if the regularization helps to counteract some of the negative overfitting effects of having higher capacity with more layers. Below is a table with the converged validation accuracy for different combinations of layers and dropout rate.

LRCN	**Dropout rate 0**	**Dropout rate 0.2**	**Dropout rate 0.4**
1 Layer	73.1	73.9	74.0
2 Layers	73.1	72.4	73.2
3 Layers	71.5	71.6	72.0

Table 2: Effect of varying number of LSTM layers and dropout rates in LCRN model

	Adadelta	**SGD**	**Adam**
C3D	84.2	82.1	81.3
LRCN	74.1	73.9	73.0

Table 3: Effect of varying optimizers on C3D and LRCN models using $3 \times 3 \times 3$ kernels and $3 \times 3 \times 1$ kernels, respectively, and dropout rate 0.2

We also explored the performance of different optimizers for the LRCN architecture. The results are summarized in Table 3. We experimented with Adadelta, SGD, and Adam again to see if the effect is the same for this model as it is for the purely convolutional model. We try these optimizers with controlled settings for other hyperparameters, by using one LSTM layer with dropout rate 0.2.

4 Discussion

Our goal in this project was to apply deep networks to the problem of hand gesture video classification. We explored the C3D and LRCN architectures to compare the performance of purely convolutional models and convolutional-recurrent models. We experimented with several different hyperparameter settings to find the model with the best validation accuracy.

For the convolutional C3D architecture, we experimented with different kernel sizes (height, width, and depth) and optimizers. Table 1 shows that higher kernel heights/widths lead to lower validation accuracy for this dataset. When it comes to kernel depth, the opposite holds true. We see that higher kernel depths corresponds to higher validation accuracy. This makes sense because a larger kernel depth means the 3D-CNN can extract more details in the time-domain at each convolution step, and therefore can take advantage of more of the information that the video has to offer. Table 3 shows that Adadelta performs the best, while Adam performs the worst. Overall, our best score with the C3D architecture was 84.2 percent, using the model with a $3 \times 3 \times 3$ kernel with Adadelta optimizer.

For the convolutional-recurrent architecture, we experimented with different numbers of layers, dropout rates, and optimizers. Table 2 shows that more layers tends to lead to a lower validation accuracy. The opposite is true for dropout rate, as increasing it increases validation accuracy. This makes sense because increasing the number of LSTM layers increases the representational capacity of the recurrent model, and so the network is more prone to overfitting. The higher dropout rates help in counteracting this, because increasing the dropout rate helps to regularize and thus reduces overfitting. Among optimizers, we see that Adadelta performs the best once again. The best score with the LRCN architecture was 74.1 percent, using the Adadelta optimizer, 1 LSTM layer, and dropout rate 0.2.

It is interesting to note that the LRCN architecture performs worse than the C3D in general for our video data, despite the supposed superiority of recurrent layers when dealing with temporal data. The reason for this could stem from our usage of a fixed kernel depth 1 in our LRCN experiments. We perform 2D convolutions on each frame in order to preserve all timesteps of the data before the recurrent layer, which could be hindering the effectiveness. In the future, we could specifically look into the use of higher kernel depths for the convolutional portion of the LRCN (and thus lower sequence length going into the LSTM layer), because it is possible that our recurrent layer could complement the use of a 3D-CNN phase rather than just a 2D-frame-by-frame CNN phase.

Based on our results above, we can see that for the Jester dataset, the best performing model is the C3D architecture with Adadelta optimizer and kernel dimensions 3. We attained a validation accuracy that we are happy with, considering we are training a 27-way classifier. It is possible that it could be even better since we haven't fully explored the hyperparameter space of these architectures. Specifically, we didn't get a chance to run all our "optimal" hyperparameters together, for example the C3D architecture with Adadelta optimizer, kernel depth 4, and kernel height/width 2. With more time and resources (training 1 epoch took around 50 minutes), we could explore more combinations of our hyperparameters, along with other variables as well. Through the project, we learned that 2D CNNs can be effectively extended to classification of videos by using a 3D kernel. We also learned a lot about how the hidden feature vectors produced by CNNs and RNNs are transferrable, which can also be applied to the classification of spatiotemporal data.

5 Advice for future DL students

- Make sure to set up your environments for assignments and projects as soon as you can. Running into problems with setup near the deadline can be disastrous.
- Prior to starting the project, think about what hardware resources are available to you because the experiments can vary significantly depending on hardware limitations.
- Always read literature prior to trying to implement models as you can waste a lot of time testing different models.
- Like we have learned, it may not be possible to explore the entire hyperparameter space, so pick and choose which ones are most important to vary according to the literature.
- *For instructors:* it was not very helpful that there was a significant coding assignment due a week before the final project was due. It might be beneficial to plan the assignment due dates ahead of time so that students can spend more on the final project towards the end of the semester. In addition, it would be helpful to have the students think about the final project at the beginning of the semester so that students have time to receive more meaningful feedback from instructors.

References

[1] He, K., Zhang, X., Ren, S., Sun, J. (2016). *Deep residual learning for image recognition.* In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 770-778).

[2] Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton. *Imagenet classification with deep convolutional neural networks.* Advances in neural information processing systems. 2012.

[3] Tran, D., Bourdev, L., Fergus, R., Torresani, L., Paluri, M. (2015, December). *Learning spatiotemporal features with 3d convolutional networks.* In Computer Vision (ICCV), 2015 IEEE International Conference on (pp. 4489-4497). IEEE.

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

[5] Wu, Z., Wang, X., Jiang, Y. G., Ye, H., Xue, X. (2015, October). *Modeling spatial-temporal clues in a hybrid deep learning framework for video classification.* In Proceedings of the 23rd ACM international conference on Multimedia (pp. 461-470). ACM.