Human Hand Gesture Classification with 3D Convolutional Residual Neural Network
Individual Report

Introduction:

The objective of this project was to research, develop, and train a neural network to classify video clips of human hand gestures using the knowledge and material obtained in this Deep Learning course. The team consisted of two members, and shared work consisted of:

- Researching interesting applications of neural networks and current topics of interest in the field, researching possible datasets for use, evaluating their complexity and scoping the overall project
- Configuring each of our respective virtual environments (Google Cloud Platform) to load, pre-process, and conduct training of the network
- Determination of best network architecture through research of existing publications and code
- Implementation and modification of pre-processing, network architecture, and training scripts to evaluate model by adjusting parameters, adding and subtracting layers, and recording results
- Collaboration on report detailing methodology, results, and conclusions

I enjoyed exploring the applications of neural networks, the problems that can be solved with them, and the importance of having large-enough data and knowing how to pre-process it for input into a network.

Description of Individual Work

After honing in on an appropriate dataset, I was able to find a recent publication and repository containing code relevant to our problem of classifying videos, which utilized a 3D ResNet and CNN architecture to classify videos, using number of images as a third dimension (depth), in addition to the dimensions of each image (height and width). This resource was found via the 20-BN website. After loading, unzipping and splitting the data, much time was spent training different models using different parameter values, epochs, layers, and kernel shapes, and max versus average pooling. The team split up these training runs and bounced ideas off one another, paraphrasing concepts to aid each other's learning. We had many discussions about ResNet architecture, the best way to preprocess the data (given inconsistent size across depth and width dimensions), and how to narrow down the vast amount of videos available through the 20BN-Jester dataset into a dataset we could feasibly manage given computation time and power. We discussed the patterns we were seeing, particularly when the model seemed to be overfitting, and chose ideal parameter values accordingly, such as batch size, learning rate, and number of epochs. Personally, most time was spent training the network with different parameter values, adding and subtracting layers in the network architecture, and playing with kernel shape. These training experiments were split up – for example, some of the experiments I performed consisted of experimenting with different optimizers and max pooling versus average pooling – and we both checked each other's results when they seemed

incorrect or unlikely. My contributions also include to the final report/analysis and development of the presentation.

Training Algorithm Description

3D ResNets ("residual networks") were used as the architecture of the network. ResNets are a type of convolution network. The key feature of ResNets is that at the end of each "block" (each block consists of two sets of convolution, batchnorm, and pooling layers), the input is added back into the output. This is done to help minimize the effects of the "degradation problem" or "vanishing gradient" problem, in which the network stops learning if it is too deep.

3D ResNet models were evaluated on training and testing accuracy, by experimenting with batch size and learning rate parameters as well as dropout, kernel size, and average versus max pooling in the network architecture. The occurrence of overfitting was also assessed by monitoring the batch loss and testing versus training accuracy. Results of training and test accuracy as well as computational time and power were recorded for the tuning of each parameter, allowing us to combine results at the end of the analysis and choose a best model. A final model was selected based on the ideal parameters selected through experimentation, and overfitting was avoided by stopping training after test accuracy stopped increasing. Each training run took over an hour, which limited the number of experiments that were attempted. A best network architecture and parameter values were selected based on testing accuracy in combination with computation time and power.

These experiments revealed ideal values for parameters like batch size and learning rate, and led to interesting discussion on a variety of hyperparameters. The network with the highest test accuracy used an Adam optimizer with a learning rate of 0.001, a batch size of 50, and was trained for 10 epochs. It classified the test samples with 59.10% accuracy. At the conclusion of the analysis, my main take-away was that the adjustment of parameter values and layers alerted us to overfitting, which allowed us to improve the model, but that adjustment of parameters did not contribute to significant performance increases in terms of test set accuracy.

Experimentation

Experiments on several different parameters were run, including: batch size, learning rate, dropout layers, number of kernels, and pooling layer shape and type. The number and types of experiments that could be run were limited by available time (each training and testing run of 10 epochs took over an hour) and computation power. For example, using a deeper network with more than four residual blocks caused training to take an unreasonably long time. Batch sizes of more than 300 could not be run on these machines due to GPU memory limitations. Allowing the networks to train for more epochs would likely have resulted in increased accuracies, but there was not enough time to allow for this.

To detect overfitting, training and testing accuracies were examined and compared. Training set accuracy that is much higher than test set accuracy was considered evidence of overfitting. Test accuracy that decreased from one epoch to the next was also a sign of overfitting. In these instances, training was stopped after the decrease in test accuracy. Based on these things, dropout and weight decay were tried as solutions to reduce overfitting.

Results:

A baseline network was trained for 10 epochs with a batch size of 100, using an Adam optimizer with a learning rate of 0.001. This achieved a test accuracy of 52.31%. This result was used as a point of comparison for most of the other experiments.

I first wanted to see how the network was learning with and without adding the input back into the output at the end of each residual block. While the networks used in this project are somewhat deep, there is other research[1] in which dozens of residual blocks are used in a single network. ResNets are designed to minimize the effects of the degradation problem which occur with deep networks, so this experiment would answer the question of whether using a ResNet was necessary in the first place. During experimentation, the network did not learn very well without the addition of the residual input at the end of the block.

Block Architecture	Test Set Accuracy	Notes
With Added Input	52.31%	Test accuracy continued to increase up to 10th epoch
Without Added Input	16.47%	Training was stopped after test accuracy decreased between second and third epochs

Table 1: Test accuracy with and without the added residual at the end of each residual block. Each network used a batch size of 100 and an Adam optimizer with learning rate of 0.001, training for up to 10 epochs

It seemed that the added residual at the end of each block was necessary for the network to learn effectively. Without it, test accuracy stagnated quickly. Even with our network that had “only” four residual blocks and a total of nine convolution layers, the degradation problem seemed to have an impact on the network’s ability to learn.

The use of a max pooling layer following the initial convolution layer, paired with an average pooling layer at the tail of the final residual block achieved the best test set accuracy. Use of max pooling after the initial convolution allows for retainment of the most important information. Much of this data is likely not important, so it makes sense to use max pooling to only retain the most important outputs.

Average pooling in the final pooling layer picks up more nuanced features that would have been discarded with max pooling, all while reducing the data for the final linear layer. The combination of these pooling layers, in the described order, resulted in the following test set accuracy values.

Pooling Layer Types	Test Set Accuracy
Initial: max pool; Final: average pool	52.31%
Initial: average pool; Final: average pool	40.53%
Initial: max pool; Final: max pool	48.44%

Table 2: Test accuracy using different types of pooling layers. There are two pooling layers in the network: one following the initial convolution layer, and one following the final residual block (see Figure 2). Each network used a batch size of 100 and an Adam optimizer with learning rate of 0.001, training for up to 10 epochs

The network was trained and tested using six different batch sizes. Based on this experiment, 50 seemed to be the optimal batch size. It has been shown[2] that, generally speaking, using smaller batch sizes tends to lead to better accuracy in deep learning problems that use SGD and other related optimizers.

However, a batch size of 10 was too low for the network to learn much at all. This may be because there are 27 classes, so each time the weights are updated, the updates are done in a way learns only about the classes contained in that batch of 10 samples. The small batches may also be causing the weights to oscillate in a way that does not lead the network towards a minimum.

Batch Size	Test Set Accuracy
10	9.99%
25	54.25%
50	**59.10%**
100	52.31%
150	53.59%
300	48.32%

Table 3: Test accuracy using different batch sizes. Each network was trained using an Adam optimizer with a learning rate of 0.001, for up to 10 epochs.

Networks were trained and evaluated using several different learning rates. From the results, it seems that the ideal learning rate is certainly larger than 0.0001 and smaller than 0.025. With a learning rate of 0.0001, it is obvious based on the train and test set accuracies that the network was drastically overfit. It seems to have learned the specific properties of each individual video in the training set, rather than learning the true underlying function that differentiates each class from the others.

When the network was trained with a learning rate of 0.025, it did not learn very well either. The large weight updates may have caused the weights to move past minima points or oscillate around them, rather than gradually approaching these points.

Learning Rate	Test Set Accuracy	Notes
0.0001	41.83%	Extreme overfitting - train accuracy of 99.12%
0.001	52.31%	
0.005	47.90%	
0.01	48.30%	
0.025	20.77%	

Table 4: Test accuracy using different learning rates. Each network was trained using an Adam optimizer with a batch size of 100, for up to 10 epochs.

After all of the experiments, the network that produced the highest test accuracy had a batch size of 50, with an Adam optimizer, a learning rate of 0.001, using an average pool layer with a 2x3x5 kernel and no dropout layer. This network achieved a test accuracy of 59.10%

after 10 epochs. Based on the difference between training and testing accuracies, there may be some overfitting. It is less extreme than in some of the other networks that were trained, though. Test accuracy continued to increase throughout the 10 epochs, which is encouraging and suggests that it would have continued increasing had it been allowed to train for more epochs.

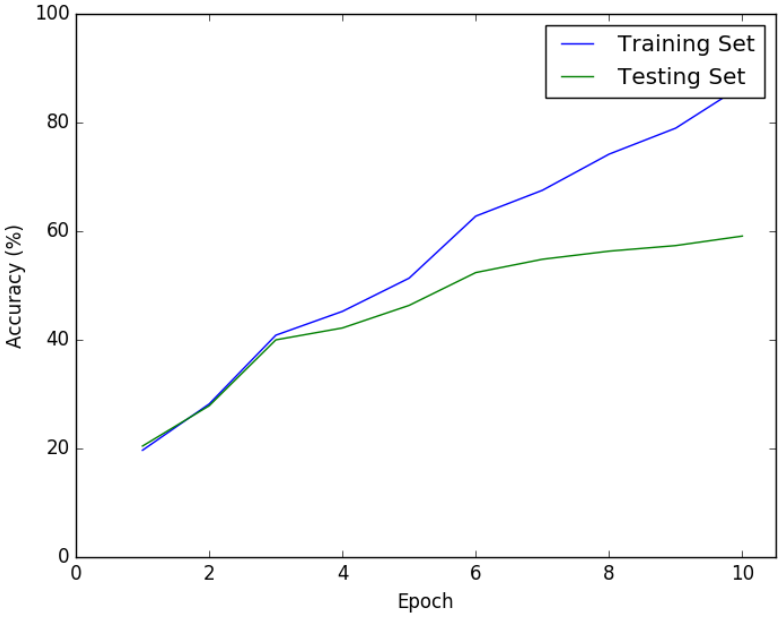

Figure 3: The training and testing accuracy by epoch for the highest performing network

The confusion matrix below shows which classes were misclassified as other classes. Several interesting misclassifications are highlighted below in red. For example, “Pushing two fingers away” was misclassified as “Pushing hand away” more often than it was classified correctly.

	Doing other things	Drumming Fingers	No gesture	Pulling Hand In	Pulling Two Fingers In	Pushing Hand Away	Pushing Two Fingers Away	Rolling Hand Backward	Rolling Hand Forward	Shaking Hand	Sliding Two Fingers Down	Sliding Two Fingers Left	Sliding Two Fingers Right	Sliding Two Fingers Up	Stop Sign	Swiping Down	Swiping Left	Swiping Right	Swiping Up	Thumb Down	Thumb Up	Turning Hand Clockwise	Turning Hand Counterclockwise	Zooming In With Full Hand	Zooming In With Two Fingers	Zooming Out With Full Hand	Zooming Out With Two Fingers
Doing other things	281	11	10	3	15	18	2	1	3	3	2	2	2	1	7	2	3	4	3	19	10	4	2	7	7	1	3
Drumming Fingers	7	123	0	0	3	0	0	1	2	3	0	0	0	0	0	1	1	0	0	3	1	0	1	1	3	1	1
No gesture	2	0	135	0	0	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	2	0	0
Pulling Hand In	3	1	0	64	24	8	0	1	1	1	2	0	1	0	3	6	0	0	13	2	0	2	0	1	1	2	1
Pulling Two Fingers In	6	1	0	8	104	9	0	1	1	2	0	0	1	2	1	3	1	0	9	0	1	0	1	0	0	3	1
Pushing Hand Away	7	0	0	5	2	121	0	0	0	2	0	0	0	0	2	0	0	0	0	1	2	0	0	1	0	0	0
Pushing Two Fingers Away	10	4	0	4	32	42	26	0	1	0	3	0	0	1	1	2	0	1	8	1	7	1	0	1	0	2	2
Rolling Hand Backward	1	6	0	1	2	0	0	43	71	0	1	0	0	0	0	5	0	0	1	1	0	0	0	1	0	0	0
Rolling Hand Forward	1	7	0	0	0	0	0	10	120	0	0	0	0	0	0	2	0	0	3	0	0	0	0	0	0	0	0
Shaking Hand	8	4	0	0	3	8	0	0	0	99	0	0	0	0	7	1	0	0	3	2	0	0	4	8	0	5	0
Sliding Two Fingers Down	3	1	0	2	6	3	2	1	2	0	95	0	0	6	3	30	0	0	3	1	6	0	0	0	1	0	1
Sliding Two Fingers Left	2	3	0	1	0	0	0	0	0	0	0	90	5	0	0	1	37	1	0	0	0	0	0	0	0	0	0
Sliding Two Fingers Right	6	1	0	1	2	2	0	0	0	0	0	1	106	0	0	0	4	35	0	0	1	2	2	0	1	0	0
Sliding Two Fingers Up	5	0	1	3	5	8	4	0	1	0	13	0	0	58	4	5	0	0	30	0	4	0	0	1	1	1	2
Stop Sign	10	1	2	0	6	27	0	0	0	6	0	0	1	0	104	2	1	0	0	1	4	1	0	2	4	2	0
Swiping Down	7	1	0	7	14	14	2	0	1	2	14	1	0	0	5	75	1	1	10	2	1	0	0	0	2	2	2
Swiping Left	2	3	0	0	2	4	0	0	0	1	0	19	1	0	0	1	116	2	0	0	0	0	1	0	0	0	0
Swiping Right	1	0	1	1	0	0	0	1	2	0	0	0	36	0	0	0	6	87	0	0	0	1	0	0	0	0	0
Swiping Up	1	4	0	9	7	15	1	0	2	1	5	0	1	9	1	8	1	0	92	1	1	0	0	2	0	1	0
Thumb Down	13	3	0	0	0	4	0	0	3	3	0	0	0	0	0	0	0	0	0	2	119	0	0	0	3	0	3	1
Thumb Up	16	1	0	0	24	15	8	0	0	1	3	0	0	2	14	0	0	0	4	0	62	0	0	2	3	1	1
Turning Hand Clockwise	12	12	1	3	3	6	1	0	0	6	0	1	2	0	0	0	0	1	3	0	0	34	18	3	1	2	0
Turning Hand Counterclockwise	7	7	0	2	5	5	1	0	0	7	0	1	0	0	1	0	2	0	2	0	0	12	48	2	0	4	1
Zooming In With Full Hand	3	8	1	2	5	8	2	0	0	4	0	0	0	0	8	0	0	1	0	0	1	10	3	83	18	7	2
Zooming In With Two Fingers	9	4	1	0	4	2	0	0	1	1	3	0	1	2	5	0	0	0	0	0	4	4	1	14	79	5	6
Zooming Out With Full Hand	4	8	0	0	3	3	1	0	0	6	2	0	0	0	2	2	0	0	0	2	0	1	2	8	1	83	25
Zooming Out With Two Fingers	10	6	0	0	5	1	0	0	0	3	2	0	0	0	1	1	1	1	0	1	7	1	2	2	17	12	86

Table 8: Confusion matrix for the highest performing network. Each row is an actual class and each column is a predicted class. Correct classifications are highlighted in green, several notable misclassifications are highlighted in red

Summary and Conclusions:

So much of this project was spent learning about ResNet architecture and the tedious hyper-parameter tuning involved when training a network. I learned about the purpose of max pooling versus average pooling, and made inferences about why those particular layers might be working best in their max-then-average pool order in this network architecture. Evaluation of network performance based on test accuracy was conducted by adjusting batch size, learning rate, experimenting with inclusion of dropout layers, pooling layer types, and final pooling layer kernel shapes. One of the most important findings was that the inclusion of residual input at the end of each residual block did actually promote network learning. Without it, the network seemed to encounter the so-called “degradation problem” that can plague deeper networks. This was evident when batch loss quickly plateaued and accuracy on the testing set stagnated, events that both go away when the residual input is included.

Batch size experimentation was also important in this analysis, as performance varied widely depending on batch size - batches that were too small or too large led to decreased accuracy. 50 was determined to be ideal. Similarly, there was a range of learning rates that produced reasonable accuracy, but learning rates that were too low (0.0001) led to extreme overfitting while learning rates that were too high (0.025) made the network unable to learn effectively.

Overall, most adjustments to the model and network architecture led to the discovery of overfitting rather than actual improvement on the test set. Ideally, the experiments would have led to much better performance. The network that produced the highest test accuracy had a batch size of 50, with an Adam optimizer, a learning rate of 0.001, using an average pool layer with a 2x3x5 kernel and no dropout layer. This network achieved a test accuracy of 59.10% after 10 epochs.

This performance was encouraging considering the limited time and computational resources. I am curious and look forward to exploring whether this network architecture has even more potential in terms of classification accuracy, if more computational resources were available so that a deeper network could be trained with more epochs, using the entirety of the dataset.

References:

1. Kensho Hara, Hirokatsu Kataoka, Yutaka Satoh: Learning Spatio-Temporal Features with 3D Residual Networks for Action Recognition
2. Nitish Shirish Keskar, Dheevatsa Mudigere, Jorge Nocedal, Mikhail Smelyanskiy, Ping Tak Peter Tang: On Large-Batch Training for Deep Learning: Generalization Gap and Sharp Minima. arXiv:1609.04836.
3. https://towardsdatascience.com/residual-blocks-building-blocks-of-resnet-fd90ca15d6ec
4. http://ruder.io/optimizing-gradient-descent/index.html

Appendix:

1. https://github.com/kenshohara/3D-ResNets-PyTorch
2. https://github.com/amir-jafari/Deep-Learning/