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

Introduction:

In this project, 3D ResNets were used to classify videos of human gestures. The dataset needed to be downloaded, decompressed, and split into train and test sets. A lot of data preprocessing was also required in order to standardize the durations and widths of the videos. Research was done on potential architectures and algorithms. Then, code was written to train and test the networks. Many runs of the networks were done using a variety of different hyperparameters (more details on this are in the group report). Finally, the reports and presentation were made based on the entire process and results.

Description of individual work:

I did research on different architectures and algorithms that could be used to solve this problem. This included reading papers and blog posts (including references 1-3 as well as others).

I wrote code for decompressing and splitting the data (this is in preprocess/unzip_and_split.sh), preprocessing the data (as part of the DataLoader; this is in network/utils.py), and the ResNet network code in network/resnet.py, some of which was repurposed from this GitHub repository.

Training was done collaboratively - we each suggested several parameters to try varying, and then split up the training runs between us. But, in particular, I did the training for the experiments that changed the final pooling layer kernel shape, as well as some of the different batch size experiments, dropout layer experiments, and the run that did not add the input back into the output in the residual blocks (more information on these experiments can be found below and in the group report).

Results:

We both contributed to performing the experiments and writing about them in the group report. Some of the ones that I contributed most to are below:

The first experiment that was run was to train the network 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 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.

In the baseline network without a dropout layer, there did appear to be some overfitting, based on the large difference between train and test accuracy. Therefore, trying to use a dropout layer to reduce this overfitting was a logical next step. A dropout layer was inserted just before the final linear layer. Increasing dropout probability caused the training accuracy to decreased significantly and the test accuracy to decrease slightly, as shown in Table 5 below.

The fact that training and test accuracy were more similar when the dropout layer was used indicates that it may have reduced some overfitting. However, it did not actually lead to increased test accuracy.

Dropout Probability	Test Set Accuracy	Notes
None	52.31%	Training accuracy reached 76.90%
25%	50.91%	Training accuracy reached 68.99%
50%	50.28%	Training accuracy reached 63.30%

Table 5: Test accuracy using dropout layers with different dropout probabilities. Each network was trained using an Adam optimizer with a batch size of 100 and learning rate of 0.001, for up to 10 epochs. The dropout layer was added just before the final linear layer.

The shape of the kernel in the average pooling layer following the last residual block was another point of interest in this project. In other ResNet implementations[1], this pooling layer used a kernel with the same shape as the input to this layer - so that the pooling layer effectively just took the average of the inputs and outputted them as a 1x1x1. This means that the final linear layer must predict 27 different outputs (one for each class) based on just this single scalar, plus a bias term.

Intuitively, it would seem that a smaller pooling kernel (and therefore a larger output of the pooling layer) would make the task of this final linear layer much easier. However, this experiment did not show this. Several different pooling kernel shapes were used and the 3x4x6 kernel, which outputs a 1x1x1 scalar to the final linear layer, produced the highest test accuracy. Apparently, passing more data to the final layer is not necessarily better; a lot of meaningful information can be contained in this single scalar. This is an interesting finding that may be of interest in future studies.

Final Pooling Layer Kernel Shape	Test Set Accuracy

2x3x5 (outputs a 2x2x2)	52.31%
1x4x6 (outputs a 3x1x1)	46.36%
3x4x6 (outputs a 1x1x1)	55.37%

Table 6: Test accuracy several different kernel shapes for the average pooling layer following the last residual block (see Figure 2). Each network was trained using an Adam optimizer with a batch size of 100 and learning rate of 0.001, 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.

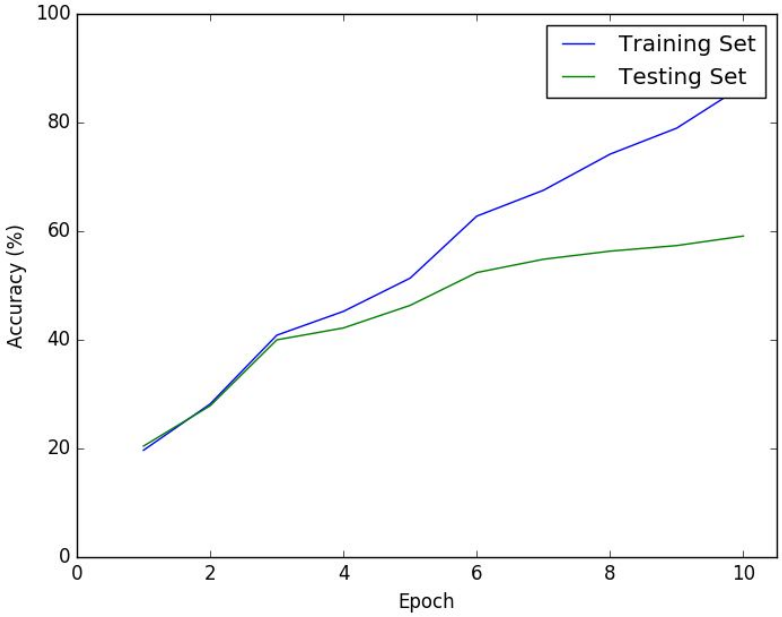

Figure 3: The training and testing accuracy by epoch for the highest performing network

The accuracy table below shows how accurately each class was predicted. There is a lot of variation among accuracies for different classes. “No gesture” was predicted very accurately (96%), while “Pushing two fingers away” was only predicted correctly in 17% of cases.

Actual Class	Class Accuracy
Doing other things	66%
Drumming Fingers	81%
No gesture	96%
Pulling Hand In	47%
Pulling Two Fingers In	67%
Pushing Hand Away	85%
Pushing Two Fingers Away	17%
Rolling Hand Backward	32%
Rolling Hand Forward	84%
Shaking Hand	65%
Sliding Two Fingers Down	57%
Sliding Two Fingers Left	64%
Sliding Two Fingers Right	65%
Sliding Two Fingers Up	40%
Stop Sign	60%
Swiping Down	46%
Swiping Left	76%
Swiping Right	64%
Swiping Up	57%
Thumb Down	77%
Thumb Up	39%
Turning Hand Clockwise	31%
Turning Hand Counterclockwise	45%
Zooming In With Full Hand	50%
Zooming In With Two Fingers	54%
Zooming Out With Full Hand	54%
Zooming Out With Two Fingers	54%

Table 7: The rate of correct classification for each class. For example, 81% of instances of the "Drumming fingers" class were correctly classified as "Drumming fingers".

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	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:

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, but the results were interesting nonetheless. 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. It also worth considering that there are 27 different classes, so a random guesser would only predict the correct class about 5% of the time. It is likely that the methods used in this project could be used to train a network that performs much better if more computational resources were available so that a deeper network could be trained with more epochs, using the entirety of the dataset.

Another thing I learned in this process is just how big truly big data is. Our full dataset was over 40 gigabytes uncompressed, and even with an 8-GPU instance we were unable to actually use most of it because it was too big to train with in a reasonable amount of time. I also got more experience and learned more about how to code convolutional neural networks using PyTorch. I had to use some advanced Python features like subclassing in order to code the dataloader and ResNet architecture, so that was also a good learning experience.

In the process of writing the code, I had to follow the data as it moved through the layers in order to make sure I understood how the network worked so I could better explain it. This gave me an opportunity to learn more about how meaningful information is extracted (and less useful information removed) from data in a convolution network.

In addition to being able to train using more data, epochs, and layers, there are also some other things that could be improved about this project. Some different methods could have been used for preprocessing. For example, instead of zero-padding (in this case, zeroes represent gray pixels) images less than 176px wide, the images could have been “stretched” horizontally to make them 176px. Some images were only 100px (so there were 38px of gray on each side), while others were already 176px. This could have made it difficult for the network to

extract meaningful information from the outer areas of the videos (because for some videos there is no meaningful information in those areas).

Percentage of Original vs. Borrowed Code:

Most of the code I wrote for this project is original. All of the data preprocessing code in network/utils.py is original. Most of the network code is network/resnet.py is based on code from [4], but a lot of that is modified. Based on the provided formula, about 26% of code I wrote for this project is copied from the internet.

References:

1. http://openaccess.thecvf.com/content_ICCV_2017_workshops/papers/w44/Hara_Learning_Spatio-Temporal_Features_ICCV_2017_paper.pdf
2. https://www.cv-foundation.org/openaccess/content_cvpr_2015/papers/Liang_Recurrent_Convolutional_Neural_2015_CVPR_paper.pdf
3. https://towardsdatascience.com/residual-blocks-building-blocks-of-resnet-fd90ca15d6ec
4. https://github.com/kenshohara/3D-ResNets-PyTorch