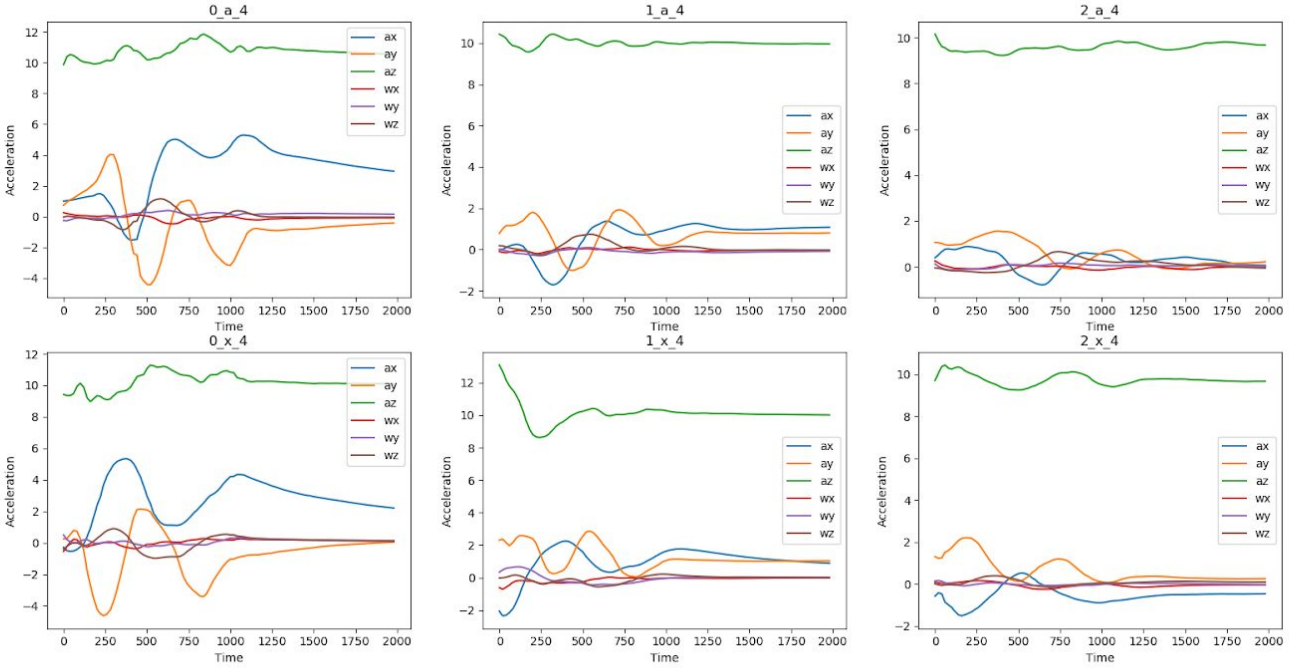
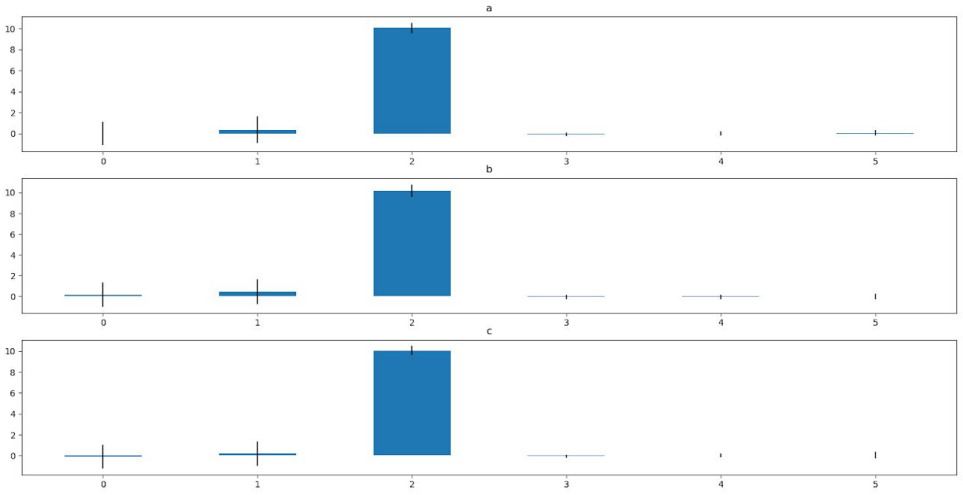
2

2.2



1. The above figures use ‘a’ in the top row and ‘x’ in the bottom row. It’s clear that there are distinguishing patterns in the graphs. In particular, y axis accelerometer acceleration is telling. In the ‘a’ there are initial bumps in the positive direction at around 250 ms, negative acceleration at 500 followed by another positive and positive at 750 and a negative at around 1000. On the other hand the x has negative bumps at 250, a positive at 500 and then another dip at around 750. There is also sometimes an earlier bump even before that.
2. These observations make sense. The a had 4 changes in direction (the initial tail, the circular motion and the ending tail). The x consisted of two slashes in opposite direction explaining the changes in acceleration direction after the initial bump.
3. The receptive field should convolve across time to classify maxima and minima. Some of the minima and maxima for each letter vary within time so having large pooling layers or large strides for the fields may be necessary to capture large amounts of times and finding the maxima/minima across these times. For example, the initial maxima for student 2’s ‘a’ in the above graph is at around 375 while student 0’s ‘a’ is at around 250. Large receptive fields, large stride lengths and large pooling kernels may be necessary. Alternative, using multiple convolutional and pooling layers with smaller kernels will ideally capture these vast features through large amounts of overlap between the kernels.

2.3



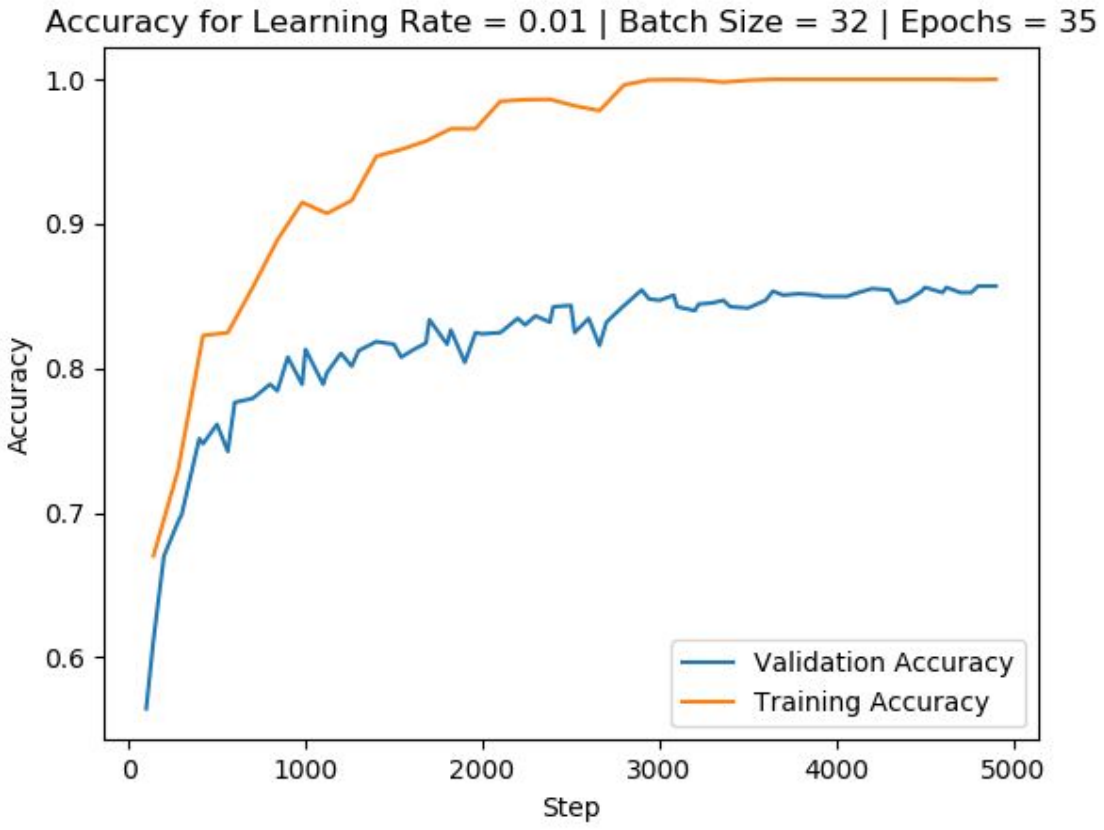
1. No not at all, the means are close enough that they're difficult to distinguish. The means and standard deviations do not preserve time variance which are key to determining the spatial structure of each letter. Removing this data makes it extremely difficult to classify as you only have the average magnitude.
2. No, the features we're left with are few and their values do not vary enough. You don't preserve any structure so you only have the average magnitudes to work with when the acceleration over time is important for recognizing the shape of each letter. You're effectively losing data by condensing it to these categories and it hurts distinguishability.

2.5

1. A test set is necessary for "sanity" check purposes. Your training set may be improperly run through, and as your validation set likely runs on the same code, it may be erroneously evaluated as well. The validation set is also usually sampled from the same data as the training set so it's possible that set is biased, whereas the testing set may not be. In general having a second round of validation is good.

3.5

The maximum validation accuracy achieved was 85.6%.



4.2

My initial network involved three layers of kernel size 7 with max pooling layers in between and a 3 layer fully connected layer. I used only ReLU as an activation function, and applied it between each layer. I believed large kernels with pooling would do a good job at extracting large amounts of features while filtering out noise, by virtue of the fact that the large features would bully out small variations in the dataset. This network achieved an accuracy of 77% while being fairly small and robust, but it clearly began overfitting after a while and reached a local minima rather than a global one. I thus evolved my network heavily into the following form:

Convolutional Layers	The number of convolutional layers I had was changed dramatically over the course of the neural network. I realized that the dataset we had was fairly limited in features with only 600 total per image, so I decided to optimize the amount of features. I decreased the size of the kernels and increased the number of layers as follows: ● Changed the kernel size from 7 to 3 ● Doubled the number of convolutional layers ● Increased the number of channels The additional layers that were added merely slotted into the other layers with 1:1 channel mapping. The convolutional part of my network evolved from (channel in, channel out) = (6,16), (16,24), (24,32) to (6,16), (16,32), (32,64), (64,64), (64,128),

	(128,128). Notable was the addition of 1:1 channel mapping which I theorized to be the network's "generalizers", serving to only create those large features that I hoped would generalize the features to represent those large curves reflecting each letter's shape. I also removed bias save for the last layer. Bias appeared to have a negative effect on other layers, and I believe because the error propagation backwards during backpropagation and gradient descent itself has some issues that
Pooling	I continued to use max pooling but removed it between my generalizers so as to
Fully Connected Layer	Fully connected layers appear to have all but a marginal effect on the accuracy of the network. They do not preserve structure like convolutions so I found it best avoid using more than one layer. ● Single linear layer mapping to the 26 outputs, no full connections
Activation Functions	I changed my activation function from ReLU to Leaky ReLU. Leaky ReLU prevents the occurrence of 'dead neurons' by allowing a small slope to exist below 0. Thus if a weight is initialized at 0 it can climb back up to a positive value if gradient descent wills it so.
Batch Normalization	I found the logic of normalizing each batch similar to how one normalizes the inputs fairly effective logic in improving my results. ● Batch normalization added after every convolutional layer
Dropout	I did not find dropout a very useful technique. I also read online that there's no real purpose to using it with batch normalization so I didn't bother using it.
Optimizer	While optimizers like Adam were very useful for prototyping my model due to its fast convergence, I found that using Stochastic Gradient Descent was far superior for generalization. SGD with momentum = 0.9 was in fact perfect for preventing erratic behaviour while getting past local minima. Nesterov was found detrimental: Nesterov is optimal assuming that the data is nearly quadratic at the input, which in our case is unlikely to be true. The data given was extremely variant. ● Stochastic Gradient Descent with momentum = 0.9 was implemented
Loss function	Cross Entropy Loss was used as the problem is multicategorical. Other functions appeared to be inferior in most cases.
Hyperparameters	Hyperparameters were mainly chosen through testing. It was generally found that there were large amounts of local minima surrounding the data so using a high learning rate and small batch size was necessary for training to find a global minima.

	• Batch size: 32 • Learning Rate: 0.01 • Epochs:
Additional Data Processing	Surprisingly enough, removing values such as the z axis acceleration and the gyroscope values actually decreased the accuracy significantly. Humans most likely naturally move their hands up and down even when they're trying their hardest to keep straight. An interesting result overall.
Training Set-Validation Set Size Changes	While final training was done on an 80/20 split, the model was tested using a 50/50 split for sanity purposes. The accuracy of the validation set only decreased by 1% with the change. The 80/20's higher accuracy however meant that it was considered best for training as a result.

5

1. Feedback
    a. I spent around 18 hours on assignment 3.
    b. I found optimizing my model challenging, especially because I have a stone tablet for a processor
    c. I enjoyed the task of optimizing the neural network and the competitive part of the assignment
    d. I found some of the instructions confusing. The constant changes to the assignment were also quite annoying and time consuming to deal with.
    e. I liked building my model from scratch.