**CNN**:

A normal neural net uses passes features through a the function in each node - called the activation function - that moves attempts to act as a discriminant function and draw a line between the samples where one side is a class and the other side is everything else. This happens in every node along the way and the connections between each node in one layer and all the nodes in the following layer are each separately weighted. At the output, the number of nodes equals the number of classes. Once a set of features goes through feedforward, the network compares the actual output to the expected output and then alters all the weights through backpropagation. An RNN is similar, but differs in that it feeds data back into itself, giving it a sort of memory. MatLab’s CNN is specifically designed for images, so it could not be successfully used for the time-series data used in this project. Instead, the we manually extracted the features and then fed those into the RNN and use that to classify each of our samples.

**MaxEpochs** - we set this to 600 through a series of trial and error. An epoch is complete when all the training set has been fed into the network once. Multiple epochs are required in order to converge to a solution ( effectively moving the discriminant lines little by little until they are in good places that minimize error )

**MiniBatchSize** - this is just the number of samples per iteration. This was was to 150 through trial and error.

**InitialLearningRate** - the higher this is, the more aggresively the network tries to converge to a solution. That said, we set it to 0.01 through trial and error, after finding that 0.001 and 0.0001 tended to get stuck at 50% accuracy and take ~60 min to converge and often overfit - as indicated by the low accuracy when the network tried to predict the testing data.

**Hidden Layer Count** = 10, through trial and error, this seemed to converge quickly and have the highest accuracy levels

**Feature Extraction** - We extract the mean, max, variance, and RMS of each channel in both the time and frequency domains and then we normalize that set of features.

NOTE:

- The SVM uses hand-picked channels that best represent each action. They were selected based on which channels showed the highest amplitudes. This process could not successfully be automated by the time of presentation of the project.

- The CNN gets all of these attributes on all the channels, because the net requires that the number of channels be identical when fed into the network. The channels missing after hand-picking could have been filled with zeros, but this was not attempted, because the net was functional and accurate without this.

**SVM -**  After a lot of trial and error trying to code an SVM, the data was formatted so that it could be fed into the SVM, with each row of a table as all the extracted features and the final column in the row as the encoded class name ( 1,2,3…,8 ). All available variations of SVMs in MatLab were attempted. The final one used used a 3rd order polynomial kernel function and a one-vs-one design, where the SVM takes every 2-class combination of all the classes, draws a discriminant between the two classes using all the features, and then outputs which class it ‘thinks’ is more likely. After doing this for all combinations, it just picks the class with the highest number of ‘votes’ and says that is the class. The **fitcecoc** function outputs the classnames - and not the encodings - so no conversion needed to be done later.

We use 5-fold cross **validation** - this means that the data is partitioned into 5 random sets. The SVM is run in a loop where each of the five sets is used as the validation set for 5 different training/validation sessions. Then the accuracy is computed from the accuracies of these five.