

Classifying EEG Spectrograms by Phalangeal Articulations utilizing Long-term Recurrent Convolutional Neural Networks

**Robert Valencia**

Electrical and Biomedical Engineering IV (Co-op)

**Electrical Engineering 4OJ4/4OH4 Research Project**

Submission date: May 22, 2017

Supervisor: Dr. James P. Reilly

McMaster University

Faculty of Engineering

Department of Electrical and Computer Engineering

Table of Contents

[1. Introduction 3](#_Toc483229251)

[2. Theory of Operation 3](#_Toc483229252)

[2.1. Data 3](#_Toc483229253)

[2.2. Model 4](#_Toc483229254)

[2.2.1. Convolutional Neural Networks (CNNs) 4](#_Toc483229255)

[2.2.2. Recurrent Neural Networks (RNNs) 7](#_Toc483229256)

[2.2.3. Multilayer Perceptrons (MLPs) 9](#_Toc483229257)

[3. System Architecture 9](#_Toc483229258)

[3.1. Overview 9](#_Toc483229259)

[3.2. Model 9](#_Toc483229260)

[4. Requirements 10](#_Toc483229261)

[5. Application Programming Interface 10](#_Toc483229262)

[5.1. ESPA Module 10](#_Toc483229263)

[5.2. EEG Processor Module 10](#_Toc483229264)

[5.3. Converter Module 10](#_Toc483229265)

[5.4. Training Configuration File 10](#_Toc483229266)

[6. Training 10](#_Toc483229267)

[6.1. Setup 10](#_Toc483229268)

[6.2. Results and Discussion 10](#_Toc483229269)

[7. Recommendations 11](#_Toc483229270)

[8. References 11](#_Toc483229271)

[9. Appendix 12](#_Toc483229272)

# 1. Introduction

This design document contains information on the operation, architecture, API, and performance of the ESPA system, developed for the research on classifying EEG spectrograms by phalangeal articulations utilizing long-term recurrent convolutional (LRC) neural networks.

# 2. Theory of Operation

## 2.1. Data

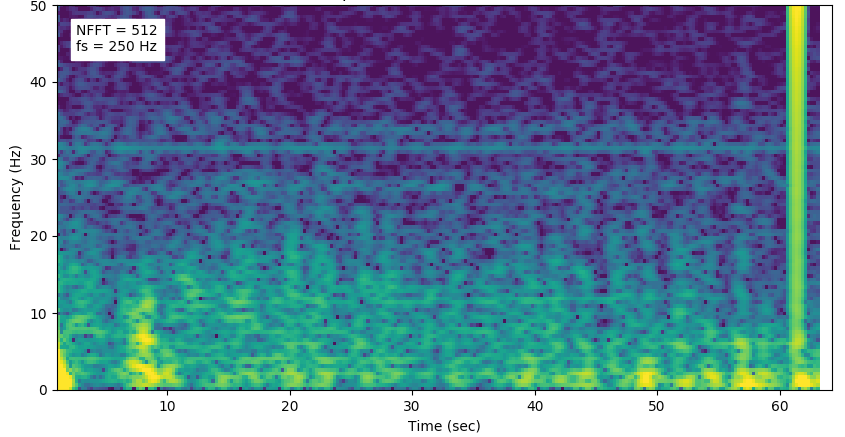


Figure Sample spectrogram from 1 channel for sustained left middle finger flexion

For the inputs, the raw data consists of EEG time signals from 8 channels. The data from each channel is cleaned by performing the following steps:

1. Trim setup and teardown data
2. Remove DC offset
3. Notch mains interference
4. Bandpass filter frequencies from 1 to 50 Hz

After cleaning the data, spectrograms are computed, then partitioned into multiple samples with a dimensionality of 250 frequency points by 50 time points. Also, depending on the training run configuration, the samples are either replicated or augmented to increase the sample size. Finally, each sample is labelled with a one-hot encoded representation of its class. For example, [1., 0., 0.] would indicate class 1 in a 3-class model.

For the outputs, arrays containing predicted probabilities for each class are utilized, where the probabilities are represented in fractional form. For example, [0.75, 0.15, 0.10] would indicate 75% probability for class 1, 15% probability for class 2, and 10% probability for class 3.

## 2.2. Model

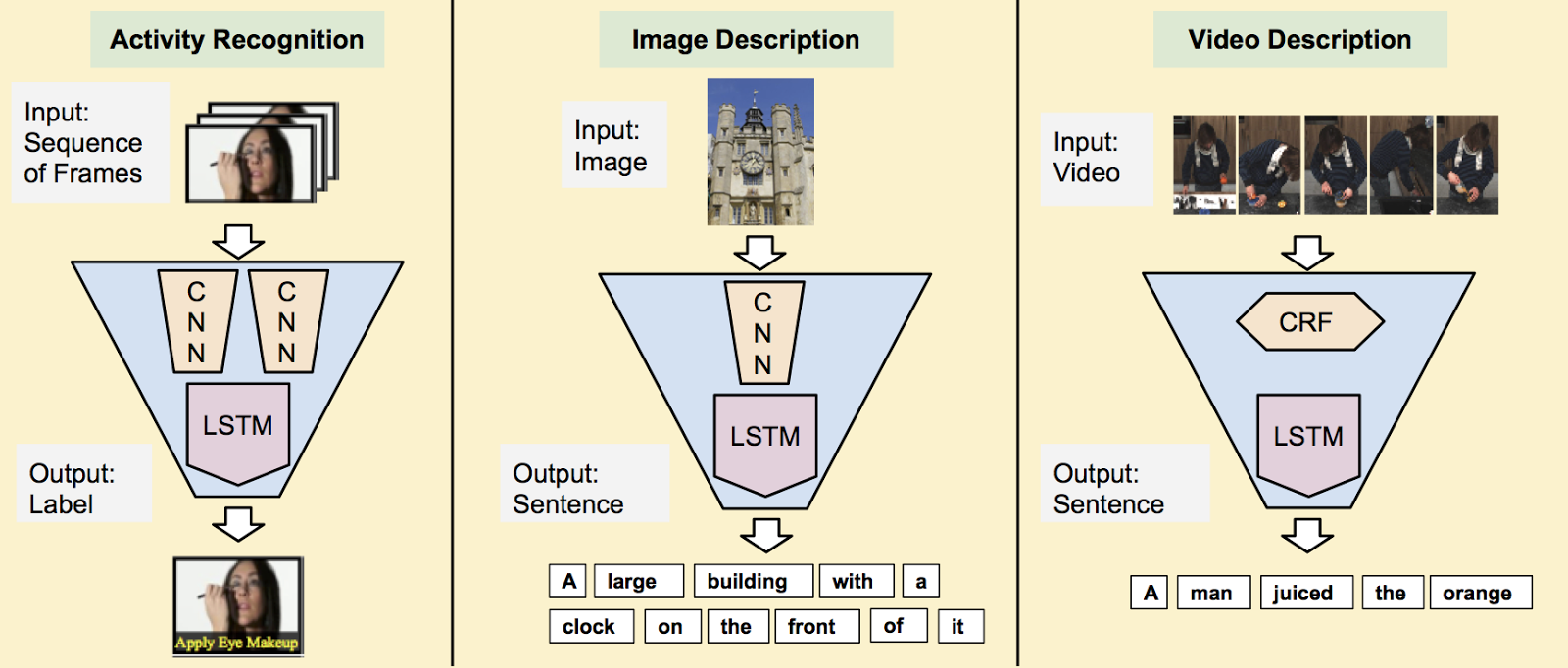


Figure Applications of LRC neural networks (Source: http://jeffdonahue.com/lrcn/)

The model is based on long-term recurrent convolutional (LRC) neural networks, a class of neural networks used for visual and sequence learning [1]. It consists of a hybrid architecture of convolutional neural networks, recurrent neural networks, and multilayer perceptrons.

### 2.2.1. Convolutional Neural Networks (CNNs)

CNNs are biologically-inspired artificial neural networks that mimic the visual cortex. In a visual cortex, there are complex arrangements of cells that are sensitive to stimuli within a restricted region known as a receptive field. This region is tiled across an entire visual field, where the cells act as localized filters for detecting spatial patterns, the response to which can be approximated by a convolution operation [2][3]:

To illustrate how CNNs work, a sample 5x5 image, its grayscale conversion, and its simplified digital representation, where 1 is the maximum value instead of 255, is shown below:

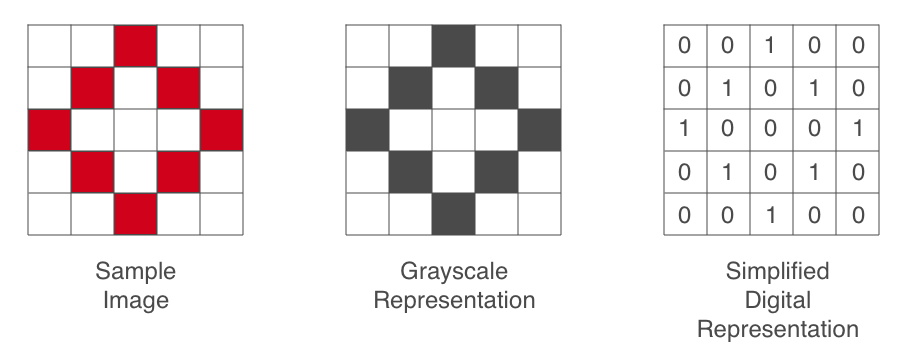


Figure A sample image, its grayscale representation, and its simplified digital representation

A sample 3x3 filter and its simplified digital representation is also shown below:

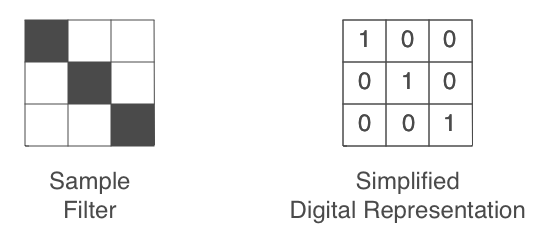


Figure A sample filter and its simplified digital representation

In CNNs, convolutional filters are tiled across an image. As the filters tile across an image, receptive fields are convolved with their corresponding visual field regions, generating an activation map. In these activation maps, regions with with a high correlation with the filter pattern have high activation values, and vice versa. In this example, the filter has a stride of 1 pixel, generating a 3x3 activation map:

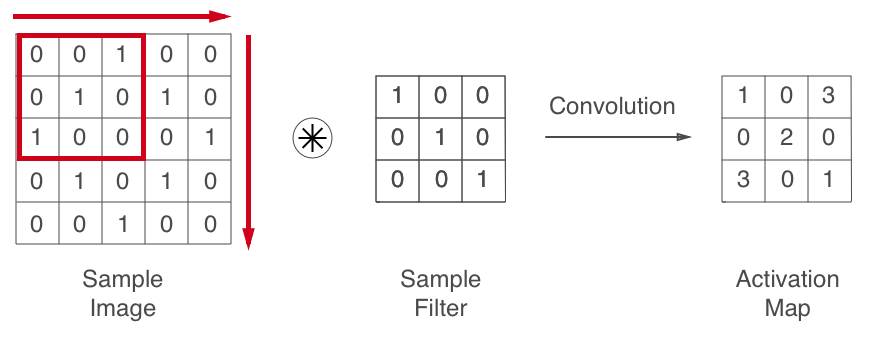


Figure Generation of an activation map by tiling a filter across an image and performing successive convolutions

These activation maps are then passed through a layer of rectified linear units (ReLUs), an activation function used to improve the network’s nonlinearity:

Finally, a pooling layer downsamples the activation maps, reducing the number of parameters. In this example, a type of pooling layer called MaxPool is used, which replaces a pool of values with its maximum values, with a pool size of 2x2 and a stride of 1 pixel:

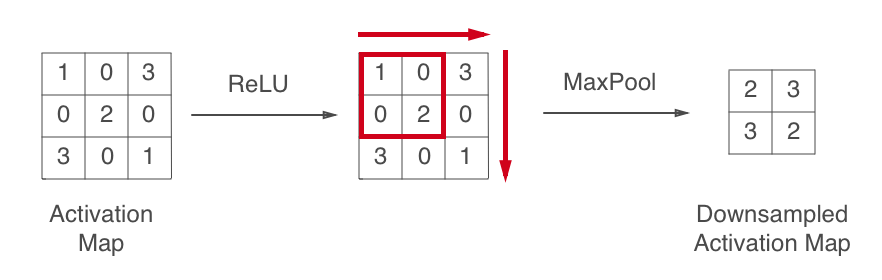


Figure Application of a ReLU activation layer and a MaxPool pooling layer to an activation map

In practical applications, multiple alternating layers of convolution, ReLU activation, and MaxPool pooling are utilized. For this model, an architecture called VGG-16 [4] is used:

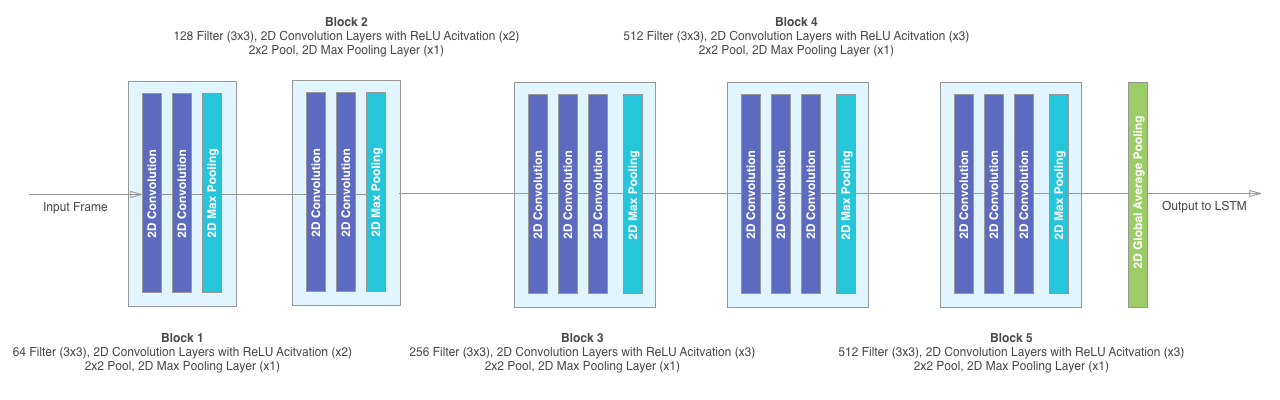


Figure General VGG-16 architecture, adapted for this model

### 2.2.2. Recurrent Neural Networks (RNNs)

RNNs are artificial neural networks that are used for detecting sequential patterns. They consist of stateful memory units that are cyclically connected. One specific type of RNN, called long short-term memory (LSTM) [5] is used in the model. LSTMs, similar to regular RNNs, consist of chains of repeated LSTM units. However, unlike regular RNNs, LSTMs are well suited for data with variable gaps between events, such as variations observed in speech due to demographic and biological variability. To demonstrate how LSTMs work, diagrams and descriptions adapted from Colah’s blog [6] and DeepLearning tutorials [7] are shown below:

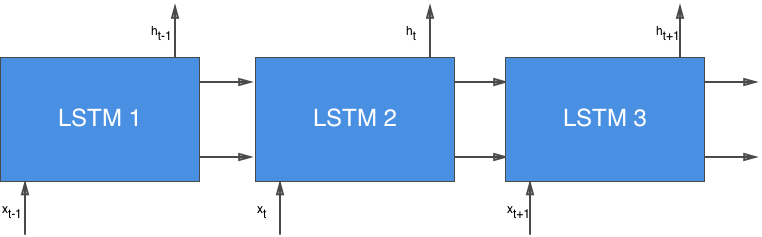


Figure Chain of repeating LSTM units

Within each LSTM unit, several operations occur, which are represented by yellow circles on the diagram below:

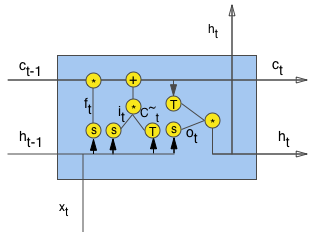


Figure LSTM operations

In the diagram above, **S** represents a logistic sigmoid operation:

**T** represents a hyperbolic tangent operation:

**+** represents element-wise addition, and **\*** represents element-wise multiplication. LSTM operations also utilize weight matrices **W**, **U**, and **V**, and bias vector **b**. First, the LSTM unit selects new data to store, which involves a logistic sigmoid layer (input gate) that selects which values to update:

and a hyperbolic tangent layer that generates new candidate values:

Next, the LSTM unit selects data to forget, which involves another logistic sigmoid layer:

The layer takes in the input, , and the previous output, , then returns either 0 or 1 for each value in the cell state , where 0 represents “forget” and 1 represents “remember”. Then, the LSTM unit updates the cell state from the old state to the new state :

This operation forgets what has to be forgotten by multiplying the old cell state with the output of the forget gate , and adds new candidate values scaled by update weights by multiplying the new cell state with the output of the input gate . Finally, the LSTM unit generates the output. First, a logistic sigmoid layer selects which values of the cell state to output:

Next, the cell state values pass through a hyperbolic tangent layer, scaling the values between -1 and 1. Finally, the outputs are multiplied, resulting in a filtered cell state:

### 2.2.3. Multilayer Perceptrons (MLPs)

MLPs are artificial neural networks that consist of fully-connected layers of nodes. They map input data into outputs via a learned nonlinear transformation, which projects input data into a space where they become linearly separable, enabling classification:

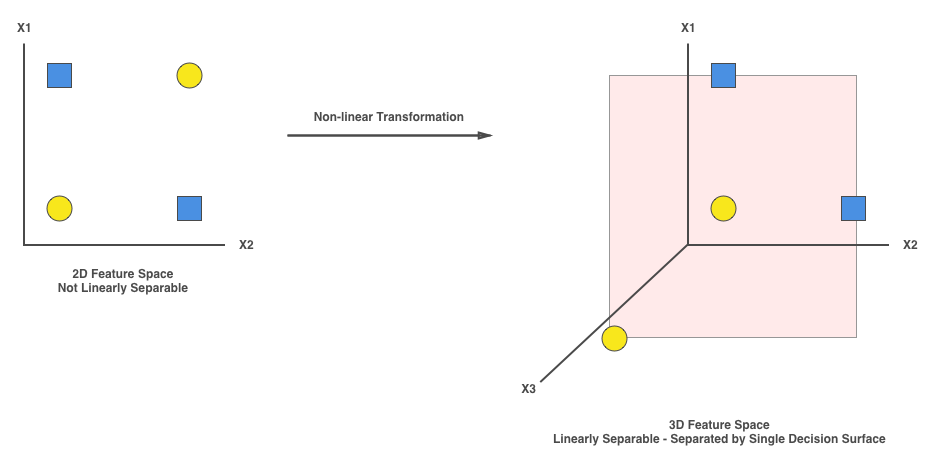


Figure Simple illustration of how projecting input data into a feature space enables classification. Two classes are represented by a blue square and a yellow circle. A decision surface is represented by a red square.

MLPs consist of 3 primary stages: an input layer, hidden layers, and an output layer. With at least 1 hidden layer, an MLP becomes a universal approximator [8]. However, in practical deep learning applications, multiple hidden layers are utilized to generate more features. In the example below, an MLP with a 2-node input layer, 3-node hidden layer, and 2-node output layer is shown:

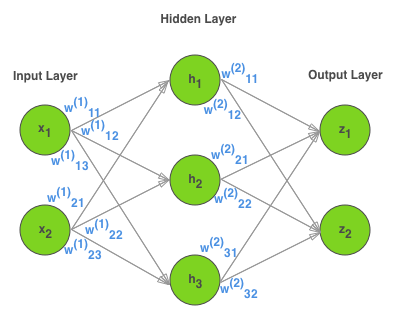


Figure Sample MLP with a 2-node input layer, 3-node hidden layer, and 2-node output layer.

In MLPs, input nodes represent input features, hidden nodes represent generated features, and output nodes represent predicted class probabilities. To make predictions, an algorithm called forward propagation is used [8]:

Where **x** is the input layer vector, **h** is the hidden layer vector, **z** is the output layer vector, **b** are bias vectors, **w** are weight matrices, **s** is the hidden layer activation function, which is set to ReLU for this model, and **G** is the output layer activation function, which is set to the softmax function for multi-class classification:

Initially, the learned parameters from **w** are randomized, resulting high error and low accuracy values. To improve accuracies, parameters are learned via the backpropagation algorithm [9][10], which trains the model on labelled data and updates parameters until a cost function is minimized.

# 3. System Architecture

## 3.1. Overview

## 3.2. Model

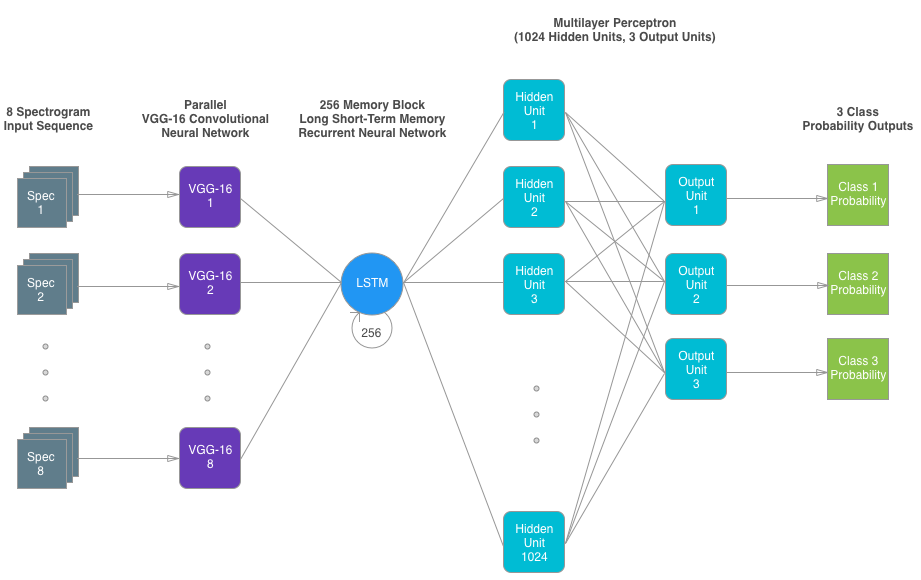


Figure ESPA model architecture

# 4. Requirements

# 5. Application Programming Interface

## 5.1. ESPA Module

## 5.2. EEG Processor Module

## 5.3. Converter Module

## 5.4. Training Configuration File

# 6. Training

## 6.1. Setup

The raw data comprises of 9, 8-channel EEG data, saved in text format, 3 for each of the following classes:

1. left index finger flexion
2. left middle finger flexion
3. left ring finger flexion

The data is then converted from text files into CSV files. Then, the data is filtered and trimmed to remove DC offset, mains interference, and setup/teardown artifacts. Next, spectrograms are calculated for each channel, each of which split into samples with 250 discrete frequency points and 50 discrete time points, and replicated 3 times to match the CNN’s input dimensions, which expects 3-colour channel RGB inputs, generating a 30 x 8 x 3 x250 x 50 training dataset. Finally, the data is saved into an HDF5 file, which is vital for on-demand loading of data as a workaround for memory resource limitations.

The following training setups were implemented, with 3 trials per setup, and 10 epochs per trial:

1. Replication, 10x sample count
2. 1% augmentation, 10x sample count
3. 5% augmentation, 10x sample count

For each setup, the data was split into the following components:

1. 60% training data
2. 20% validation data
3. 20% testing data

## 6.2. Results and Discussion

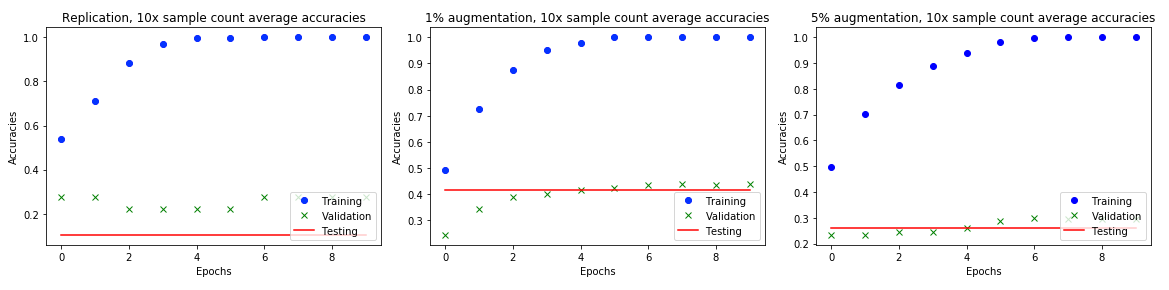


Figure Accuracies (average training, validation, and testing) across all three setups

Across all three setups, it can be observed that the training accuracies ascend to high values and plateau within several epochs, while the validation accuracies lag behind the training accuracies, which could indicate overfitting. It can also be observed that both the validation and testing accuracies are significantly higher with 1% augmentation, which could indicate that 1% augmentation provides a good balance between generating independent samples and introducing excessive noise.

# 7. Recommendations

The primary issue that has to be addressed is overfitting, which prohibits the model from generalizing to new data. Some potential solutions include:

1. Increasing the raw sample size instead of depending entirely on data augmentation
2. Implementing regularization: L1, L2, and max norm
3. Implementing dropout

Other alternative changes that could potentially improve the model’s performance include the choice of CNN architecture (e.g. ResNet, Inception), RNN architecture (e.g. Gated Recurrent Unit [GRU]), MLP architecture (deeper [more hidden layers] and wider [more neurons]). With an improved model, classification could be expanded to classify finer phalangeal articulations, given sufficient training data.

# 8. References

[1] J. Donahue, L. A. Hendricks, S. Guadarrama, M. Rohrbach, S. Venugopalan, T. Darrell, and K. Saenko, “Long-term recurrent convolutional networks for visual recognition and description,” 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2015.

[2] “Convolutional Neural Networks (LeNet),” Convolutional Neural Networks (LeNet) — DeepLearning 0.1 documentation. [Online]. Available: http://deeplearning.net/tutorial/lenet.html. [Accessed: 22-Apr-2017].

[3] S. B. Damelin and W. Miller, The mathematics of signal processing. Cambridge: Cambridge University Press, 2012.

[4] K. Simonyan and A. Zisserman, “Very Deep Convolutional Networks for Large-Scale Image Recognition,” [1409.1556] Very Deep Convolutional Networks for Large-Scale Image Recognition, 10-Apr-2015. [Online]. Available: https://arxiv.org/abs/1409.1556. [Accessed: 22-Apr-2017].

[5] [Sepp Hochreiter](https://en.wikipedia.org/wiki/Sepp_Hochreiter); [Jürgen Schmidhuber](https://en.wikipedia.org/wiki/J%C3%BCrgen_Schmidhuber) (1997). ["Long short-term memory"](https://www.researchgate.net/publication/13853244_Long_Short-term_Memory). [Neural Computation](https://en.wikipedia.org/wiki/Neural_Computation_(journal)). 9 (8): 1735–1780.

[6] C. Olah, “Understanding LSTM Networks,” Understanding LSTM Networks -- colah's blog. [Online]. Available: https://colah.github.io/posts/2015-08-Understanding-LSTMs/. [Accessed: 24-Apr-2017].

[7] “LSTM Networks for Sentiment Analysis,” LSTM Networks for Sentiment Analysis — DeepLearning 0.1 documentation. [Online]. Available: http://deeplearning.net/tutorial/lstm.html. [Accessed: 22-Apr-2017].

[8] “Multilayer Perceptron,” Multilayer Perceptron — DeepLearning 0.1 documentation. [Online]. Available: http://deeplearning.net/tutorial/mlp.html. [Accessed: 22-Apr-2017].

[9] C. V. D. Malsburg, “Frank Rosenblatt: Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanisms,” Brain Theory, pp. 245–248, 1986.

[10] D. Rumelhart, G. Hinton, and R. Williams, “Learning Internal Representations by Error Propagation,” Readings in Cognitive Science, pp. 399–421, 1988.

# 9. Appendix