DELIVERABLE REPORT

Project acronym: INPUT
Project number: 687795

D6.1, Integrated Machine Learning Software Suite
Dissemination type: DEM
Dissemination level: PP
Planned delivery date: 2016-09-30
Actual delivery date: 2016-09-30
Reporting Period: 1

WP6, Task 1, Integrated Software Suite (lead: SUPSI-IDSIA)

1 DESCRIPTION OF THE TASK

An integrated software analysis suite will be developed and compiled, containing implementations of state-of-the-art data analysis and machine learning methods, together with features for processing data in the formats generated by the project beneficiaries. During this stage, UMG-GOE will provide sample EMG data collected during previous projects (AMYO, MYOSENS) to test and evolve these algorithms and methods early on in the project, even before the finalization of the signal acquisition hardware (WP4).

2 DESCRIPTION OF DELIVERABLE

Integrated Machine Learning Software Suite.

This software suit will be used to implement all approaches which will be investigated for solving the control problem targeted in INPUT.

(see task description)

3 IMPLEMENTATION OF WORK

An integrated software suite has been developed based on the software toolkit *Brainstorm* (by SUPSI-IDSIA). This toolkit can be used by means of scripts written in the *Python* language, a collection of scripts has been developed based on the protocols prescribed by the sample data provided by UMG-GOE and for interfacing with the live-recording system. A series of tests for the INPUT tasks has been performed on the sample dataset.

3.1 SAMPLE DATA

UMG-GOE provided a sample dataset of forearm EMG recordings of 11 able-bodied subjects (ABS) and 4 amputees (AS). In all cases, a ring of 8 electrodes was applied to the forearm (see figure 1). For ABS, recordings were performed on five different days with three recording positions with a shift of ±8mm, for the AS subjects, 10 recordings were collected, where the recording position varies naturally.

Each single recording has a length of 5 seconds and contains one of 8 different movements (*No Movement – Fine Pinch – Hand Open – Key Grip – Wrist Extension – Wrist Flexion – Wrist Pronation – Wrist Supination*). The force follows a trapezoid structure (1s increase, 3s hold, 1s decrease) up to 30%, 60%, or 90% of the maximum voluntary contraction (MVC) of the forearm muscles.

Furthermore, Otto Bock provided to SUPSI-IDSIA a recording system for testing purposes intended for able-bodied subjects. This includes recording hardware similar to the one depicted in figure 1, a USB connector, and an in-house software tool to control the recordings from a standard PC. Data produced by this recording system follows the same format as the sample data described above.

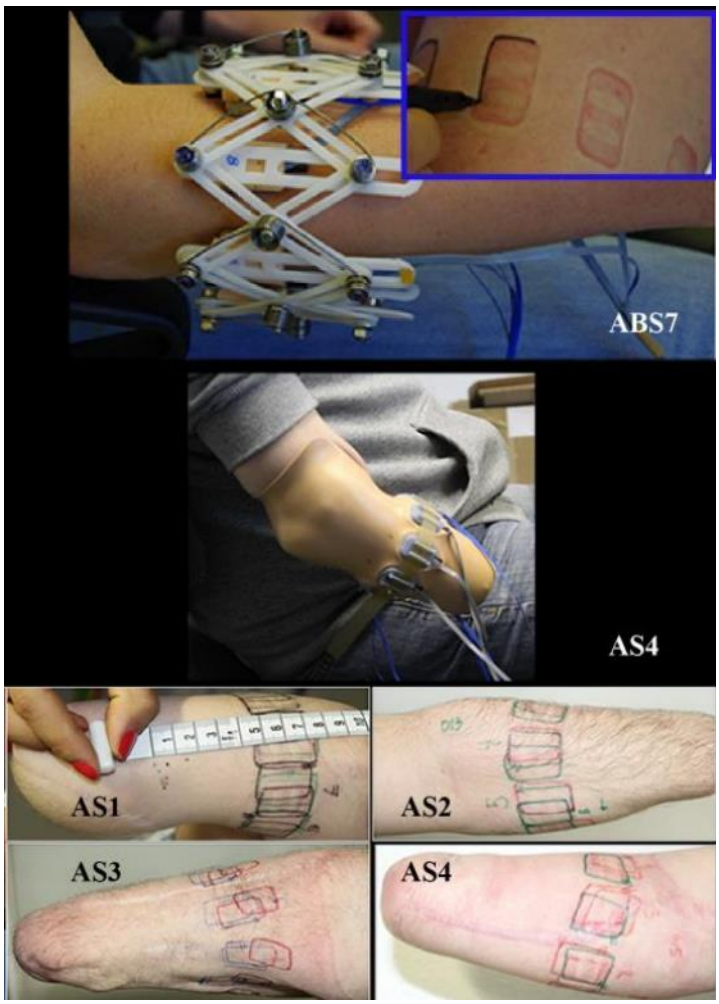

Figure 1: Photographies of recording setup for the sample dataset

3.2 THE *BRAINSTORM* SOFTWARE

Brainstorm is a framework for design, implementation, evaluation and testing of artificial neural networks. Together with the open-source experiment management tool Sacred, also developed at IDSIA, this framework makes experiments within the INPUT project very quick to set up, analyze and reproduce. Brainstorm has been developed as a general purpose software applicable to a large variety of machine learning applications. It combines lessons learned from the limitations of existing open source alternatives with new design choices which increase flexibility, portability and ease of use.

3.2.1 Modularity and Ease of Use

It is written completely in Python, a cross-platform, easy to use and widely adopted programming language in the scientific community. This ensures seamless integration with many other available data-science packages and tools. Leveraging the wealth of production quality packages in python makes it very easy to perform data-preprocessing, visualization and integration with other software. Its modular design allows for easy configuration and switching of components. For example, any experiment can be GPU-accelerated (providing up to 50x speedup), without requiring any GPU programming.

3.2.2 Recurrent Neural Networks

Brainstorm supports recurrent (fully recurrent, LSTM, Clockwork) layers from the very beginning. They are all treated as equal to feedforward (fully-connected, convolution, pooling) and can easily be combined with one another. This is a key difference from many existing neural network frameworks where recurrent networks tend to be an after-thought. For INPUT these architectures are important, since they can learn to use the temporal structure of the data. Given enough training data, recurrent neural networks could potentially learn the best ways to smooth and denoise the EMG datastream, thus eliminating the need for manually tuned post-processing.

3.2.3 Convolutional Neural Networks

Another important piece of the repertoire that Brainstorm offers are convolutional layers. Convolutions are being used with tremendous success as part of image-processing neural networks. They make use of the spatial structure of images to support the training process and help to eliminate any manual feature extraction steps. Even in speech recognition, this method has been applied to remove manual feature extraction while improving the performance. The same could apply to EMG data, thus removing the hand crafted preprocessing and potentially improving performance.

3.2.4 Flexible Architecture

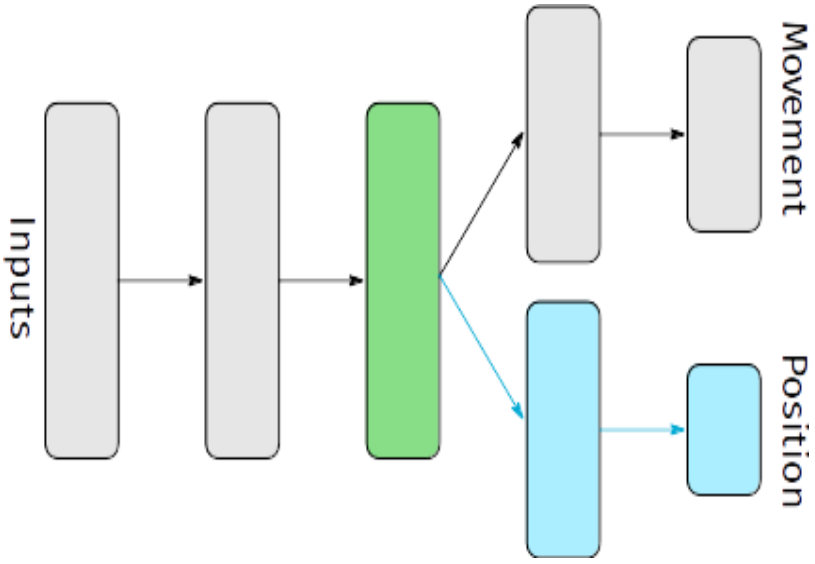

Figure 2: Flexible Multi-Task Network Architecture with Brainstorm

The modular design of Brainstorm enables the construction of a wide variety of advanced network topologies. The central abstraction in brainstorm is a layer which implements a commonly used mathematical transformation (such as convolution or pooling) and its derivative. Layers are connected to form a network defining which computations should be performed and what their parameters are.

Brainstorm has been developed to allow both simple creation of standard network topologies and fine-tuning by expert users.

This feature can be used to train a set of neural networks, one for each patient, but which share their early layers. That way the networks could learn a common feature extraction while still specializing on the high-level differences between users. Another use case is the so-called adversarial training: after a few layers the network is split into two (see Figure 1). One branch is trained for the normal classification/regression task, while the other is trained to discriminate different positions of the electrodes. By inverting the gradient flowing back from the second branch, one can encourage the features learned by the early layers to be position-independent.

3.3 Brainstorm Scripts to be used in the INPUT Project

SUPSI-IDSIA developed a Brainstorm / Python scripting framework based on the sample data provided by the project partners. The system is made available to the project partners in form of a private Github repository. Details about the scripts are reported in section 4.

3.4 Initial Results

SUPSI-IDSIA ran a series of experiments based on the sample data provided by the partners. As a proof-of-concept for the software framework, first results are reported in section 4.

4 Results

4.1 DATA PREPROCESSING AND BASELINE SYSTEM

SUPSI-IDSIA developed "helper" scripts to perform the following tasks.

- Reading of EMG data: EMG data saved in the standard data format given by the sample data can be read and processed by Brainstorm scripts. Labels for the movement, strength (%MVC), and recording positions are created from the raw data.

- EMG preprocessing: Signal preprocessing was implemented in Brainstorm following the recipe in S. Amsuess et al., *Factors of impact on the surface EMG Classification Accuracy over Days in Controls and Amputees*, unpublished draft provided by the first author. This is a baseline implementation and subject to change during the course of the project.

- LDA baseline: To ensure a common baseline for the neural networks experiments, the setup from the above mentioned paper was reproduced. Being integrated with the EMG preprocessing it comprises only a Linear Discriminant Analysis (LDA) for classification.

4.2 Brainstorm Scripts

We developed Brainstorm scripts for the INPUT tasks of movement classification and regression, where in a practical setting the latter refers to proportional, multi-DOF prosthesis control.

```
def run(NetSpec, LearnRate, TrainingSequencesPerClass, seed, MaxEpochs, BatchSize, Patience, _run, _config):
    data, targets, positions, strengths = get_data()
    data_train, targets_train, data_val, targets_val = train_val_split(data, targets)

    print('centering the data...')
    means = data_train.mean(axis=(0, 1))[None, None]
    data_train -= means
    data_val -= means

    print('preprocessing the data...')
    data_train_pp, targets_train_pp = preprocess(data_train, targets_train)
    data_val_pp, targets_val_pp = preprocess(data_val, targets_val)

    print('standardizing the preprocessed data...')
    means = data_train_pp.mean(axis=(0, 1))[None, None]
    stds = data_train_pp.std(axis=(0, 1))[None, None]
    data_train_pp = (data_train_pp - means) / stds
    data_val_pp = (data_val_pp - means) / stds

    train_getter = bs.data_iterators.Minibatches(BatchSize, default=data_train_pp, targets=targets_train_pp)
    val_getter = bs.data_iterators.Minibatches(BatchSize, default=data_val_pp, targets=targets_val_pp)

    print('create Network with spec',format(NetSpec))
    in_shape = data_train_pp.shape[2]
    net = bs.tools.create_net_from_spec('classification', in_shape, 8, NetSpec)

    trainer = bs.Trainer(bs.training.SgdStepper(learning_rate=LearnRate), verbose=Verbose)
    trainer.add_hook(bs.hooks.ProgressBar())
    scorers = [bs.scorers.Accuracy(out_name='Output.outputs.predictions')]
    trainer.train_scorers = scorers
    trainer.add_hook(bs.hooks.MonitorScores('valid_getter', scorers, name='validation'))
    trainer.add_hook(bs.hooks.EarlyStopper('validation.Accuracy', patience=Patience, criterion='max'))
    trainer.add_hook(bs.hooks.StopAfterEpoch(MaxEpochs))
    trainer.add_hook(bs.hooks.InfoUpdater(_run))

    print('starting training...')
    trainer.train(net, train_getter, valid_getter=val_getter)

    print('Best  validation result:',np.max(trainer.logs['validation']['Accuracy']))
    print('Final validation result:',trainer.logs['validation']['Accuracy'][-1])
```

Listing 1: Example Brainstorm Script (for a classification task)

Listing 1 gives an example of the main part of such a script, in this case, for a classification run (note that some functions have been left out for clarity). Data is obtained from a suitable source, split into a training and validation set, preprocessed and standardized (see section 4.1), and finally a neural network is trained. In this particular case, the validation data is tested in every training epoch, and both the best and the final validation result (in terms of recognition accuracy) are reported.

All scripts are part of the project Github repository.

4.3 Proof-of-concept Results

We report results on a classifier using a neural network with two hidden layers and 100 neurons per hidden layer, as shown in figure 1. The ReLu function is used as neural network nonlinearity.

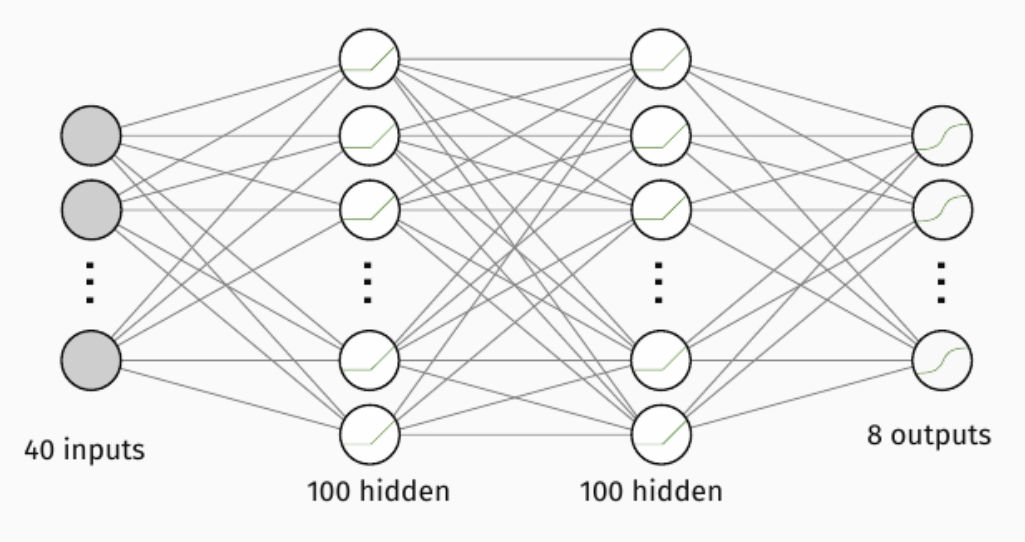

Figure 3: Neural Network Architecture for Classification Task

Resulting classification accuracies are given in figure 3, for the able-bodied subjects. We emphasize that these results are not optimized for either performance or multi-DOF control. The results are compared with the LDA baseline reported in Amsuess et al., *Factors of impact on the surface EMG Classification Accuracy over Days in Controls and Amputees*, unpublished draft. and as reproduced by us. The accuracies of the Neural Network classifier are always better, although we see that where the

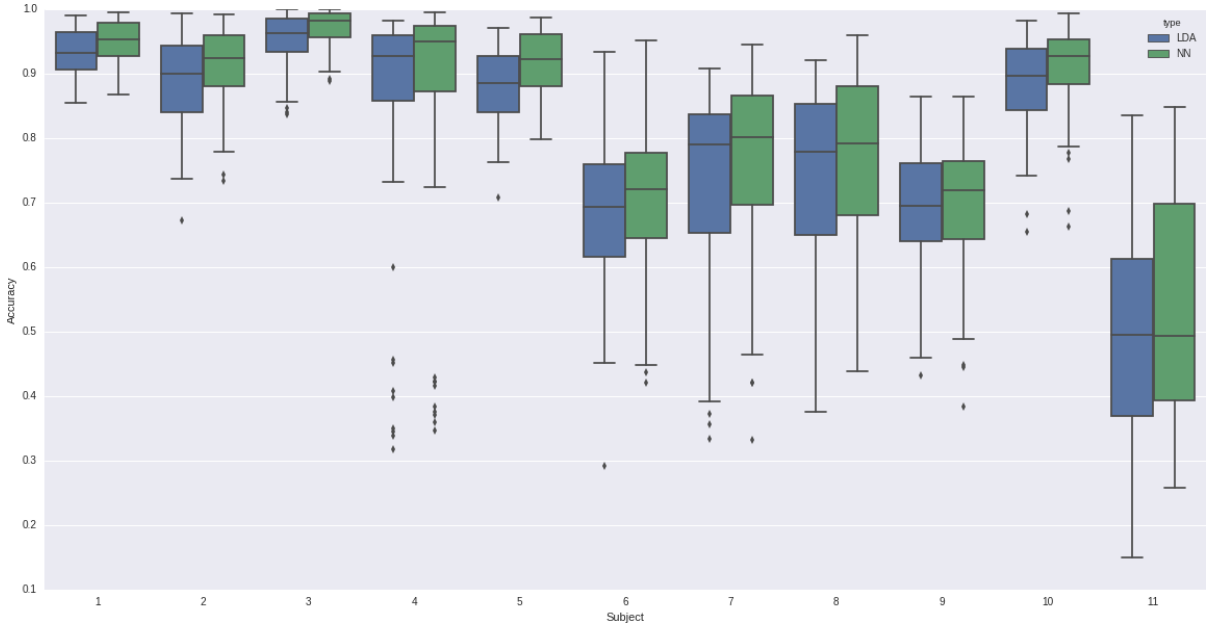

Figure 4: Recognition Accuracy of the Neural Network compared to LDA Baseline, on validation data

LDA classifier yields suboptimal results, the Neural Network classifier does not perform substantially better (figure 4).

We furthermore experimented with regression of the force (given as %MVC) on the same dataset, using the same neural network setup with only the final layer changed to a linear output layer, and the loss function changed to Mean Squared Error. We note, however, that the underlying dataset is not optimally suited for this experiment, experiments on further datasets are currently being run.

An example of force regression by class is given in figure 5 (for one subject only). It can be seen that the regression works, although the results on the validation data (right) are generally less accurate than the results on the training data (left). It is also observed that in particular the extreme case (90%MVC) tends to be underestimated, however this may be an artifact of the dataset and is expected to be mitigated in the future when datasets specifically created for regression tasks are used.

5 SUBCONTRATING

All work was done by SUPSI-IDSIA (software) respectively by the other partners as described (providing of sample data).

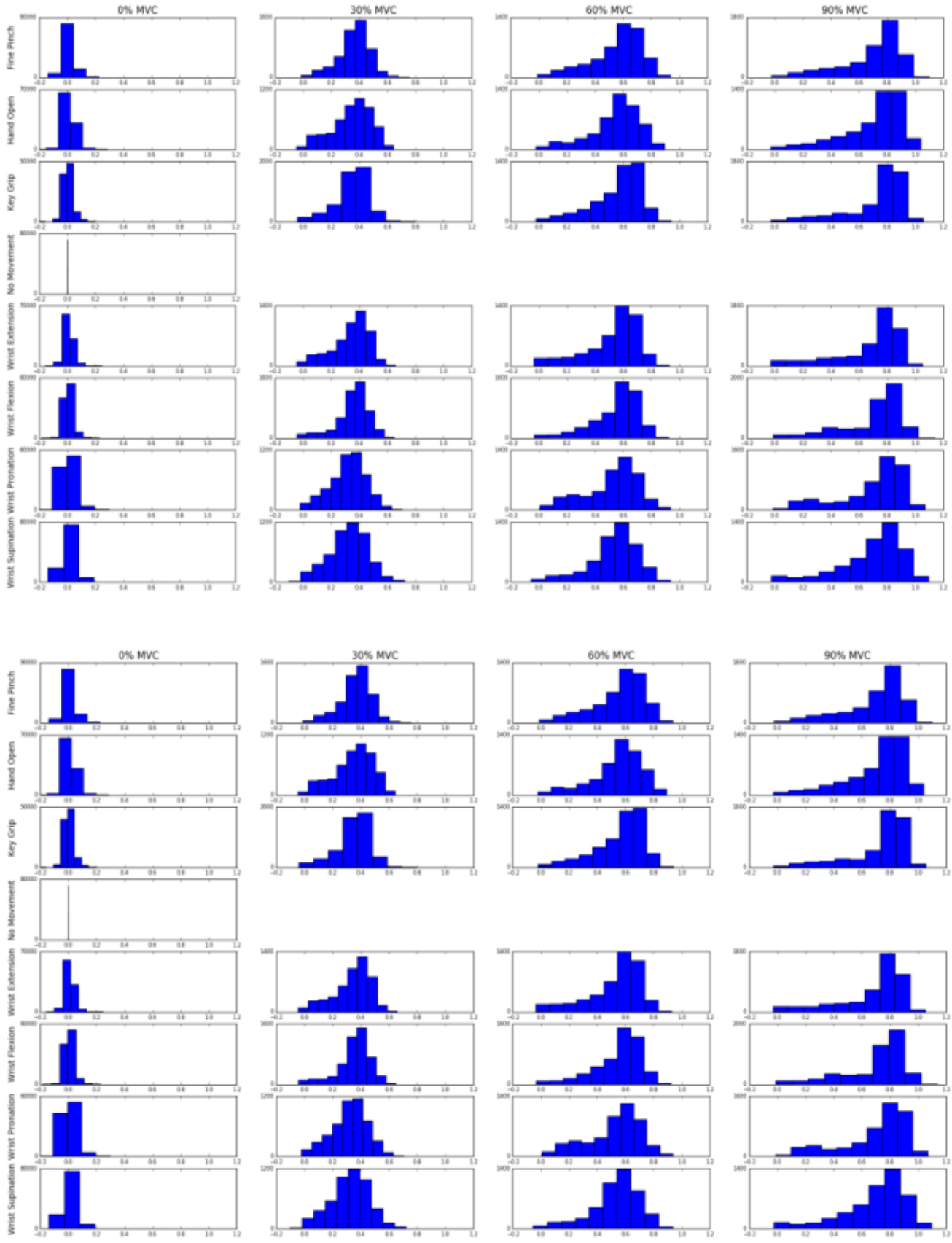

Figure 5: Example for Force Regression by class. Above: results on training data, below: results on validation data