Improving Novelty Detection with Generative Adversarial Networks on Hand Gesture Data

Miguel Simão[*], Pedro Neto[†] and Olivier Gibaru[‡]

November 5, 2018

Abstract

We propose a novel way of solving the issue of classification of out-of-vocabulary gestures using Artificial Neural Networks (ANNs) trained in the Generative Adversarial Network (GAN) framework. A generative model augments the data set in an online fashion with new samples and stochastic target vectors, while a discriminative model determines the class of the samples. The approach was evaluated on the UC2017 SG and UC2018 DualMyo data sets. The generative models' performance was measured with a distance metric between generated and real samples. The discriminative models were evaluated by their accuracy on trained and novel classes. In terms of sample generation quality, the GAN is significantly better than a random distribution (noise) in mean distance, for all classes. In the classification tests, the baseline neural network was not capable of identifying untrained gestures. When the proposed methodology was implemented, we found that there is a trade-off between the detection of trained and untrained gestures, with some trained samples being mistaken as novelty. Nevertheless, a novelty detection accuracy of 95.4% or 90.2% (depending on the data set) was achieved with just 5% loss of accuracy on trained classes.

Index terms— Collaborative Robotics, Semi-Supervised Learning, Generative Adversarial Networks, Novelty Detection

[*]M. Simão is with the Department of Mechanical Engineering of the University of Coimbra, Portugal (miguel.simao@uc.pt).

[†]P. Neto is with the Department of Mechanical Engineering of the University of Coimbra, Portugal (pedro.neto@dem.uc.pt).

[‡]O. Gibaru is with the École Nationale Supérieure d'Arts et Métiers, Lille, France (olivier.gibaru@ensam.eu).

1 Introduction

Often-times the performance of a classifier trained offline on a data set is not indicative of the online performance. This may happen due to missing elements on the data processing pipeline, such as proper data scaling. However, the major issue is the limited scope of a data set, when compared to the real problem. Independently of the resources available, a data set can not include a large portion of real-world scenarios. In those cases, there is no way of confidently predicting the performance of a classifier.

In a gesture recognition data set, there are training patterns of a predefined number of gesture classes. It is also possible to include gesture patterns that do not match any of the classes but that can occur in real-world conditions [1]. These are known as untrained gestures, novelties, non-gestures, or *others*, i.e., gestures that do not belong to the predefined classes. However, the diversity of non-gestures is almost infinite, or at least, much greater than that of the predefined gestures.

Non-gestures appear in real-world conditions on every type of interaction, particularly in human-human and human-machine interaction. They may occur due to, albeit not limited to the following reasons:

1. The user is uneducated in respect to the human-machine interface and performs out of vocabulary gestures;
2. The user is distracted and is moving in a way that does not make sense in the context of the interface, e.g., talking to somebody else;
3. The user is forced to move in response to other

elements in the surroundings, e.g., moving machines or falling objects;

4. The user is from the end of an interaction to the start of the next, i.e., Movement Epenthesis (ME).

The naive way to exclude non-gestures is to set a threshold to the output probability of a classifier:

$$\text{class} = \begin{cases} \tau, & p(y_\tau|z) \geq \text{threshold} \\ \text{none}, & p(y_\tau|z) < \text{threshold} \end{cases} \quad (1)$$

where y is the classifier's output given the feature vector z, ad p the probability distribution over the problem's classes. We have previously shown that for the most widely used classifier – ANNs –, the output probability is not a good measure of classification certainty [2]. This means that many correct classifications may have low probabilities and incorrect ones have high likelihood, as predicted by the ANN. If we were to set a threshold on the class probability using 1, many good classifications would be discarded, while an equal proportion of bad classifications would pass. Therefore, the threshold method is not effective for its purpose.

It is possible to exclude non-gestures and bad classification using context clues, such as limiting the quantity of possible outputs. However, this is a limited approach and we are interested in exploring new methodologies that may help a classifier discriminate non-gestures.

A non-gesture is an "abnormal" occurrence for the classification model, which is trained with a restricted number of classes. While non-gestures could belong to an extra class containing all of the possible non-gestures, training it is a challenge because of the lack of examples. Because of the large gesture domain, it is unfeasible to get data samples from every possible non-gesture. This problem is seldom addressed and most data sets do not include such patterns.

The problem of detecting "abnormal" patterns from a predefined number of gesture classes is generally known in the literature as Novelty Detection (ND). In this type of problem, the predefined classes have considerably more training examples, while the "abnormal" patterns are underrepresented. In [3], the authors concluded that currently, there is no optimal solution for the ND problem because if depends on the type of data and the application domain. Distance methods such as k-NN have been shown to be superior, but their computational efficiency decreases with data set size, therefore making them unsuited for larger data sets and real-time applications [4].

We propose the use of a semi-supervised methodology which uses the labelled samples of a data set and generated unlabelled samples which correspond to either gestures or non-gestures. This is an approach that is currently used in deep learning where very large data sets are required but labels are not always available [5, 6]. This is a methodology that has been used successfully on data sets for image recognition. In this chapter, we present the results of its application to hand-gesture recognition with GANs.

1.1 Generative Adversarial Networks

The name GAN describes a framework for the training of generative neural networks that was introduced by Goodfellow et *al.* in [7]. In this framework, there are two competing nets which are trained simultaneously: a generative net G and a discriminative net D. The objective of the discriminator D is calculating the probability that a sample came from the real data set rather than from the generator G. On the other hand, G is trained to produce samples that maximize the probability of D classifying them as real. D and G have competing objectives and should normally improve one another. A diagram representing a variant of this framework is shown in figure 2.2.

Current applications of GANs are essentially in the field of image to image translation, i.e., generation of new images with specified features or increased resolution [8, 9, 10, 11, 12]. Very few authors have studied applications on other domains, such as speech [13, 14] and text generation [15]. The flexibility of GANs gave rise to a plethora of network structures and training methods, of which some notable ones are: Auxiliary Conditional Generative Adversarial Network (AC-GAN) [16], Cycle-Consistent GAN (CycleGAN) [17] and Wasserstein GAN [18]. In terms of performance, it is difficult to evaluate these models quantitatively, since these are generative models and they are purpose built. Nevertheless, the generated outputs are often indistinguishable from real data in state of the art implementations.

The original GANs [7] had a discriminator D whose output was the binary classification of the source of a sample (real or generated). If the original data set was also divided in N classes, the D would also be able to classify generated data, thus appearing an extension to semi-supervised learning [16]. These are the auxiliary conditional GANs.

For the evaluation of the generated samples, an intra-class diversity measure was proposed in [16], specifically the multi-scale structural similarity (MS-SSIM). This metric aims to account only for the same features as a human would perceive, rather than calculate a pixel to pixel distance. Thus, it is more indicated for image similarity. Lack of similarity is a symptom of an important failure mode during GAN training. It happens when the generator collapses and generates a single pattern that maximally confuses the discriminator. A generator model that only outputs a single pattern is very limited and not useful, so we should avoid a collapsed generator.

In this chapter, we propose small modifications in the structure the AC-GAN:

1. A softmax layer as the second output of the discriminator;
2. A one-hot encoded second input in the generator instead of the class number;
3. Training with stochastic target vectors.

These changes aim to allow the use of the discriminator as an online classifier and the generation of samples with any given class likelihood.

2 Methods and Methodology

In this section we discuss the data pipeline for the fitting and test of the model, the architecture of the model, its training methodology and the definition of the tests.

2.1 Data Pipeline

We assume that we have available a labelled data set, which is defined as:

$$\mathcal{D} = \left\{ \left(\boldsymbol{X}^{(i)}, \iota^{(i)} \right) \ : \quad i = 1, 2, \ldots, n_{samples} \right\} \quad (2)$$

in which $\boldsymbol{X} \subset \mathbb{M}^{t \times d}$ represents the sample data of d channels (variables) and t time steps, and $\iota^{(i)} \in \mathbb{N}_0$ is the target class for that sample. For development and testing purposes, the data set is split into three subsets: training, validation and testing subsets.

The following step is feature extraction, which depends on the data set, classifier model and objectives. Generally speaking, it is defined by $\boldsymbol{f}^{(i)} = \mathcal{F}\left(\boldsymbol{X}^{(i)}\right)$, where $\mathcal{F}$ represents the extraction function and $\boldsymbol{f} \subset \mathbb{M}^{n_f}$ is the output features, which is a vector of length equal to the number of features per sample, n_f.

The features are then normalized, i.e., assuming the variables follow normal distributions, whose parameters are calculated in the training set. All of the subsets are normalized with these parameters and every new sample is normalized with these same parameters. Finally, the targets are one-hot encoded.

2.2 Stochastic Target GANs

The structure of the custom GAN is very similar to a AC-GAN [16]. There are still separated generator and discriminator networks, like in the vanilla GAN introduced in [7]. The first input of the generator G is a noise vector z with latent size l that is sampled from a normal distribution $\mathcal{N} \sim \left(\mu = 0, \sigma^2 = 1\right)$. The second input of G is a one-hot vector that represents the class to be generated, while a AC-GAN uses the class index. The generator network's structure is free, but it must be a feed-forward neural network. The structure is highly dependent on the available data and the training hyperparameters.

The discriminator D takes samples as a vector of length n_f (number of features), which can either come from a data set or the generator. These serve as input for the discriminator network, whose structure is also free, depending on the type of data. D has two outputs, being the first the validity of the input sample. The validity is a scalar $v \in [0, 1] \subset \mathbb{R}$, i.e., a real number between 0 and 1. Validity below 0.5 denotes a generated sample and if it is above or equal to 0.5, it corresponds to a real sample. The discriminator has second output, which is the classification of the sample as a one-hot vector.

The notation used in this study is explained next. The data set samples are represented by a 3-

element tuple x_r, t_r, s_r, which are the sample data, the target class and its source, respectively. These are also called real samples, as opposed to generated samples, known as x_g, t_g, s_g. x_r and x_g have the same shape as $\boldsymbol{f}$ in section 2.1 and represent the features obtained from the sample. The targets t_r and t_g, for real and generated samples respectively, are one-hot encoded vectors of the classes they represent. Assuming the classification problem has n_c classes, the one-hot vector t for class ι is a horizontal vector of size $1 \times n_c$ composed of zeros, except $t_\iota = 1$. The source scalar, s, is either 1 for real samples, or 0 for generated samples.

2.3 Model Training

The two networks are trained by Stochastic Gradient Descent (SGD) simultaneously, in two interwoven stages (D-G-D-G-D-G ...) for a number of epochs. A diagram that applies to both stages is shown in figure 1. In SGD, the weights of the model are updated according to the gradient of the model's loss on a batch of b samples. This means that instead of vectors, the inputs and outputs of the networks are matrices. In these, the first dimension corresponds to the sample and the second to the variables (features, target index, among others).

First stage: discriminator

The discriminator is trained with both real and generated samples. Given a pre-selected batch size b, $b/2$ samples are extracted from the data set, being denoted by x_r. An equal amount of samples is generated from G, x_g. These samples are generated by running G with two inputs:

$$x_g = G(z, \phi(\iota)) \tag{3}$$

The first input is the noise matrix z, which has the following definition:

$$z = \{z_{ij} \leftarrow \mathcal{N}(0,1),\ \forall i \in [1,n],\ j \in [1,l],\ i,j \in \mathbb{N}\} \tag{4}$$

where n is the number of samples to be generated (in this case $b/2$) and l is the latent dimension of the generator. Basically, a noise matrix is sampled from the normal distribution.

The second input are the one-hot vectors of the classes ι of the samples to be generated, which are sampled from a discrete uniform distribution:

$$\iota = \{\iota_i \leftarrow \mathcal{U}\{1, n_c\},\ \forall i \in [1,n] \subset \mathbb{N}\} \tag{5}$$

where n_c is the total number of classes in the problem and n is the number of samples. The one-hot encoded input is $\phi(\iota)$. For the stochastic targets, the target vector is a vector where the element $\boldsymbol{t}_{k=\iota}$ has a certain value p' between 0 and 1, while the other elements sum up to 1:

$$\boldsymbol{t}_k = \begin{cases} p', & k = \iota \\ \dfrac{1-p'}{n_c - 1}, & k \neq \iota \end{cases} \tag{6}$$

Until now, we have defined all the data required to train the discriminator. There are two analogue tuples of data: (x_r, t_r, s_r) and (x_g, t_g, s_g). The samples x_r and x_g are fed into the discriminator D:

$$(v, y) = D(x) \tag{7}$$

There are now two losses to be calculated, as seen at the end of figure 1: the validity and classification loss. The validity is a binary classification problem, thus the loss function chosen is the binary cross-entropy:

$$L_1 = -(s\log(v) + (1-s)\log(1-v)) \tag{8}$$

where L_1 is the validity loss on a sample, s is the value of sample's source (0 or 1) and v is the predicted probability of the sample belonging to the correct class, which is the first output of D.

The actual classification of a sample is a multi-class problem, so the multi-class cross-entropy is used:

$$L_2 = -\sum_{c=1}^{n_c} t_{i,c}\log(y_{i,c}) \tag{9}$$

where L_2 is the classification loss of a sample i, $t_{i,c}$ is the value of element c of the sample's target t, and $y_{i,c}$ is the output of D for the same sample.

The final loss is a weighted average of L_1 and L_2. Finally, the discriminator's weights are updated.

Second stage: generator

While the discriminator is trained with both real and generated samples, the generator is trained

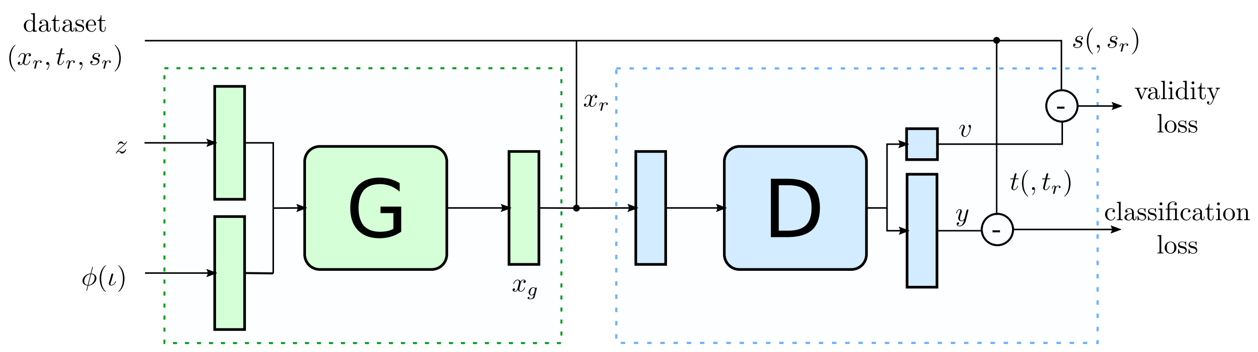

Figure 1: Diagram representing the training process of the custom GAN.

without real samples. The process is the same as shown in figure 1, but the data set is not used.

The noise z and indexes ι are sampled according to (4) and (5). Analogously to discriminator training, a batch of b samples x_g is obtained from the generator so that $x_g = G(z_g, \phi(\iota_g))$. We then calculate the validity (8) and classification (9) losses with the discriminator. Hence, the loss function of the discriminator is also used with the generator. However, the discriminator weights are frozen during this stage, so only the generator's weights are updated to minimize these losses.

2.4 Classification Decision

A trained discriminator can be used to classify new samples. However, the output class provided by the discriminator is rarely taken as the final classification. There is often a decision method that provides the final classification, possibly context-based information, such as data from other sensors.

Most of the neural network classifier models have a final layer which is a softmax transfer function. This function provides a probability distribution over the possible classes. This is the second output of D shown in (3). The probability of a given sample x belonging to class i is given by:

$$\boldsymbol{y}_i = p\left(i \mid \boldsymbol{x}\right), \quad \text{for } i = 1, 2, ..., n_c \tag{10}$$

where n_c is the number of possible classes.

Given the probability distribution $\boldsymbol{y}$, it makes sense to set a threshold τ on $\boldsymbol{y}_i$ so that if its value drops below a pre-defined value, the output class is disregarded as *others*:

$$\text{output class} = \begin{cases} \text{class } i, & \max\ \boldsymbol{y} \geq \tau \\ \text{others}, & \max\ \boldsymbol{y} < \tau \end{cases} \tag{11}$$

This definition introduces the problem of determining an adequate value for the threshold.

The use of this method results in more false negatives than using no threshold (or $\tau = 0$). As a consequence, increasing the threshold yields lower recall of the classes i. Therefore, it is possible define a threshold value such that the recall does not decrease further than, e.g., 5 or 10%. In most applications, false positives are worse than false negatives, the exception being in critical conditions such as a request for an emergency stop. These cases are rare and should be specially handled to increase safety.

3 Tests and Results

This section describes the test methodologies followed in this chapter. We are interested in evaluating the performance of both the generator and the discriminator. The generator should find patterns in the data set samples and create new samples based on those patterns.

3.1 Result Validation

All tests use the same data split: 60% for the training set, 20% for validation and 20% for testing. The split was fixed at the beginning of analysis, so that it is not optimized for proposed methodology and no data leakage occurs. We consider this hold-out split to be better than a k-fold cross-validation split in regard to deployment-oriented analysis. On one hand, a k-fold method requires the classifier to be trained k times, thus having a training time roughly k times larger than a hold-out split. On the other hand, GANs are remarkably difficult to train and it is highly unlikely that the same hyperparameter set will allow the GAN training process to converge in all folds.

The training set is used to train the classifier and does not include any sample of non-gestures. The purpose of the validation set is to optimize the hyperparameters of the classification model, i.e., neural network structure, added noise, normalization, among others. Additionally, the model fitting process is controlled by the model loss that is calculated online on the validation set, in order to prevent over-fitting on the training data. This set does not include any non-gesture sample, so that there is no leakage of these data into the fitting process. Finally, the test set is used to test the generalization capability of the model and is only used when the training and validation sets provide desirable metrics. Therefore, the model is not optimized for the test set and the metrics calculated on it should provide a good measurement of the model performance in other conditions.

3.2 Data Sets

We tested the presented methodology on two data sets: the UC2017 Static and Dynamic Hand Gestures data set [19] and UC2018 DualMyo data set [20]. In the experiments, we denote the index 0 to the network abstractions for the first data set (GAN0, D0, G0) and 1 for the second (GAN1, D1, G1).

The UC2017 data set contains static and dynamic gesture samples captured with a data glove and magnetic tracker. The library contains 24 static gesture classes with samples obtained from eight subjects with a total of 100 repetitions for each of the 24 classes (2400 samples in total). The classifier is trained on 19 classes and the remaining 5 were set aside to be used as the *others* class (novel patterns).

There are no particular features extracted from these data. The networks are trained with raw data, which includes the hand's joint angles provided by the data glove and the hand's pitch, sensed by the tracker. Thus, the features chosen are simply a subset of channels of the available data. Finally, the features are standardized by $x'_i = (x_i - \bar{x}_i)/s_i$, where x'_i is the standardized value of feature i, x_i is the value of the feature, $\bar{x}_i$ and s_i are the mean and standard deviation of the feature in the training set. The validation and test sets are standardized by these same means and standard deviations.

The UC2018 DualMyo data set comprises 8 classes of patterns with 110 repetitions each. Class 7 was set aside to become the class *others*, or novel patterns. This means that the classifier is trained on classes 0 through 6, and tested on all 8 classes. This class was selected because it is not trivially separated from the others in an unsupervised manner.

Data samples are matrices $\boldsymbol{X} \in \mathbb{M}^{t \times d}$, where the length t is 200 frames and the second dimension d consists of the 16 Electromyography (EMG) channels. The feature extraction function chosen $\mathcal{F}$ is the standard deviation of the sample along time, i.e., one standard deviation per channel. Therefore, the feature vector extracted from each sample $\boldsymbol{f}^{(i)}$ is a vector of length equal to the number of channels. This feature is often proportional to muscle contraction strength, thus providing a muscle activation map around the forearm.

3.3 GAN Structure

The discriminator and generator networks may take many shapes, under the presented framework. Furthermore, there is no method to initialize a network structure for a given problem. Generally, it depends on the size of the data set, type and quality of the data and number of features, among others. The initial structure is found by random grid search on a varying number of layers and respective nodes in large steps. The network's performance is evaluated on the validation set. When a good structure is found, it is then optimized by manually fine-tuning the number of nodes, transfer functions and applying generalization aids. In this work, we fixed the structure of the GAN's networks for all experiments.

The structure of the generator is shown on the left side of figure 2. The depth of the network was kept at two fully-connected (dense) hidden layers with 256 nodes each. There are two inputs, the

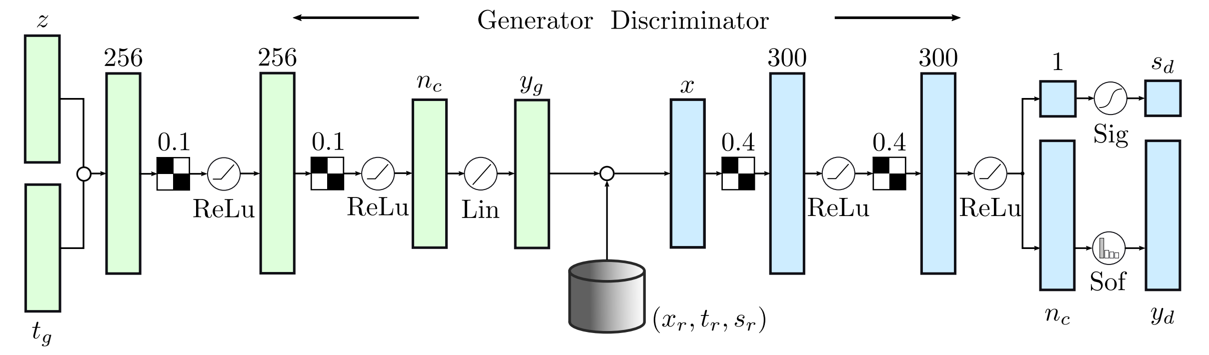

Figure 2: Structure of the GAN used on the UC2018 DualMyo data set for the evaluation of the proposed methodology.

first being a noise vector z sampled from a random normal distribution $z \sim \mathcal{N}(\mu = 0, \sigma = 1)$, with a latent size equal to the number of features n_f. The second input is the stochastic target vector. These are fully-connected to the first dense layer. As shown in figure 2, Gaussian noise $\mathcal{N}(0, 0.1)$ is added to each of the dense layers, followed by rectified linear units (ReLU). The ReLU activations are then normalized in the training batch, i.e., centred and scaled. Finally, the output is reduced to a vector of length n_f by a dense layer of that same size, after which a linear transfer function provides the network output. The length n_f corresponds to the number of features of a data set sample, so the output is equivalent to the features of a real sample.

The introduction of Gaussian noise layers is typically recommended in generator networks and helps increase the variance of generated samples. A failure mode of GANs is the mode collapse, in which the generator learns and outputs an unique sample. Adding noise during the training process, whether through noise layers or noisy labels, helps prevent this issue.

The discriminator is shown on the right side of figure 2. It has a single input x, which can be either generated (y_g) or real (x_r) samples, therefore having length n_f. A Gaussian noise layer $\mathcal{N}(0, 0.4)$ is added next. Following that, there is a dense layer with 300 nodes and a ReLU activation function. This layer is repeated one more time with the same parameters. To aid generalization, there is a dropout layer before the output, where 30% of randomly chosen connections are dropped in each training iteration. Afterwards, the networks splits into two branches corresponding to its two outputs. The first output, the validity s_d, has a single fully-connected node, whose activation is then transformed by the sigmoid function is order to return a value between 0 and 1. The second output is also fully-connected with n_c nodes, which corresponds to the number of classes of the problem. In this last case, a softmax activation function is applied in order to output a distribution of probabilities over the number of classes.

3.4 Training Parameters

In section 2.3, we mentioned that the discriminator D and generator G are trained separately, one after the other, since they are different networks. Therefore, they may have distinct training parameters, and the success of a GAN framework is strongly dependant of these parameters. The learning process must be balanced so that one does not learn much faster than the other, causing the mode collapse failure mode. This occurs when the generator is trained to a state where it always outputs the same value, independently of the input, thus bringing the learning process to a halt.

The networks are balanced through the tuning of the learning rates of D and G, and the relative weights of the two losses of the generator. As a reminder, the generator is trained with the losses calculated on the discriminator. These are the validity (8), and classification losses (9). If the classification losses decrease faster than the validity, the generator will focus on generating n_c different classes that are more easily separable, not necessarily resembling the original data. Therefore, we increase the validity loss weight, so that G learns to generate samples more akin to real data, rather than more separable data.

Other relevant parameters include the training momentum, number of epochs, the latent dimension of G, batch size and label noise strength. Additionally, there are the learning rates for D and G, and the loss weights of D. There are some recommended values for these parameters, but since there is no established optimization process, they were optimized by trial and error as a function of the network losses and sample visualization.

Despite the GAN structure being the for both the UC2017 SG and UC2018 DualMyo data sets, the training parameters were optimized in different directions. The networks GAN0 and GAN1 were trained for 600 and 300 epochs, respectively, while the batch size was 32 samples in both cases. The G0 and G1 latent sizes was set to 23 and 8. The stochastic target labels were sampled from $\mathcal{U}(0.9, 1.0)$. In early trials, it was set to $\mathcal{U}(0.8, 1.0)$, but the lower bound of 0.9 achieved better results in combination with the remaining parameters.

The SGD optimization algorithm followed the Adam update rule [21]. The learning rates were set firstly to the typical value of 0.0002 for both networks and data sets, but the rate for G ended up being incrementally increased to 0.001, in order to speed up the generator training. The discriminators D0 and D1 were trained with learning rates of 0.001 and 0.0002, respectively, and the momentum was kept at 0.5 in all cases. For both GAN0 and GAN1, the weight decay was set to 10^{-7} and 10^{-6} to D and G, respectively. Finally, the G validity loss weights were set to 1.1 and 1.3 for G0 and G1, respectively, while the classification loss weights were 1.0 for G0 and 0.8 for G1. The training time for this setup is about 289 seconds in a Tensorflow-based Keras implementation for GAN0 (63 seconds for GAN1). The nets are trained on a Nvidia GTX970M GPU with 6GB of memory.

An example of the GAN1 losses is shown on figure 3. The discriminator and generator losses are very close, which is expected since the loss functions are the same, despite the G1 loss being calculated from generated samples, while the D1 loss is obtained from equal amounts of generated and real samples. Both losses are still decreasing by epoch 300. However, looking at the generator loss components individually, we see that the classification loss is decreasing, but the validity loss has plateaued. This means that the network is improving the separation of the generated samples but their similarity to the real samples is not improving. The slight peaks found every 50 epochs are likely to be numerical errors caused by sampling G1 at the checkpoints that occur at the same frequency.

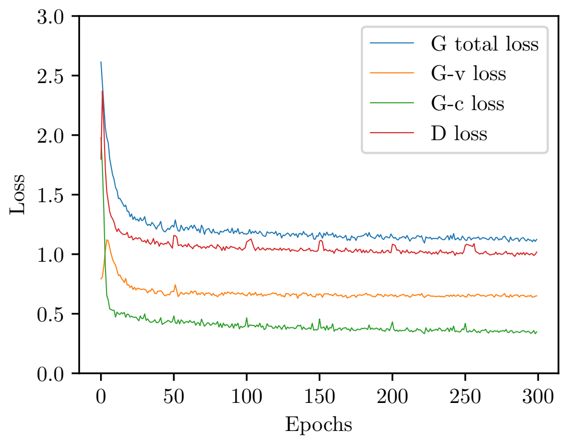

Figure 3: Plot of the training losses of discriminator D1 and generator G1 validity loss (G-v) and classification (G-c) loss components for each training epoch. All losses are monotonically decreasing and the G-v loss has plateaued by epoch 300.

3.5 Generator Performance

Firstly, we tested the quality of the samples created by the GAN's generator. The quality is determined by the similarity between generated and real samples. There are plenty of similarity measures that can be used, but since we have small feature vectors, we opted by simply using the L2 distance between samples. The concept of distance is opposite to similarity, thus the distance must be minimized to improve similarity.

Formally, we are interested in knowing the distance between two sets of samples, $\boldsymbol{X}$ and $\boldsymbol{Y}$. These sets of data are matrices of shape $(n_{samples} \times n_f)$, where $n_{samples}$ may or may not be the same. The L2 distance between the i-th sample of $\boldsymbol{Y}$ and

$\boldsymbol{X}$ is thus given by:

$$\boldsymbol{l}_i = \frac{1}{N} \sum_{j=1}^{N} \sqrt{\sum_{k=1}^{n_f} (\boldsymbol{X}_{jk} - \boldsymbol{Y}_{ik})^2} \qquad (12)$$

where N is the number of samples in $\boldsymbol{X}$ and n_f the number of features. In short, the distance between a sample $\boldsymbol{Y}_i$ and the set $\boldsymbol{X}$ is the mean L2 distance between $\boldsymbol{Y}_i$ and all of the samples of $\boldsymbol{X}$.

The following tests are presented:

1. **Data set baseline distance:**
   Mean distance between data set samples of the same class.
   Standard deviation of the distance between data set samples of the same class.

2. **Generated data distance:**
   Mean distance between real and generated samples of the same class.
   Standard deviation of the distance between real and generated samples of the same class.

3. **Gaussian noise distance:**
   Mean distance between real and random noise generated from Gaussian distributions.
   Intra-class standard deviation of the distance between real and noise samples.

Two baseline distances are established: a data set and a random noise baseline. The data set baseline establishes the ideal distance, i.e., the dispersion of the real data set. The generated data distance measures how far the generated data are from the real data, which should tend to the ideal metric when these mimic perfectly the real data. The Gaussian noise distance is the worst-case scenario that would occur if the generator were to diverge from the real data. In all cases, the distance metric is computed for sets of samples within the same class. The randomly generated data are sampled from Gaussian distributions with mean and standard deviations calculated from the real data, for each class individually.

Examples of generated and real samples for each gesture class of the DualMyo data set are shown in figure 4, where the lowest signal is represented in dark blue and the highest in yellow. A first look shows that generally, the intensity of low state signals is higher in generated samples than in real ones. Nevertheless, that is not an issue since there is still a visible gap between low and high signal states. The last class, G7 or *others*, is a class created by the generator that was not trained, therefore there is no equivalent in real samples.

While visualizing the samples provides a subjective evaluation of the quality of the generated samples, the first generator test is comprised by a similarity measurement between data set samples, generated samples and samples drawn from Gaussian distributions. The results are shown in table 1 for the UC2017 SG data set and table 2 for the DualMyo data set. The mean distance values displayed in table 1 show that by class, most of the GAN-generated samples are significantly closer to the baseline than random, the exceptions being classes 0, 2, 3 and 17. However, visualizations of the samples show that the generated samples are similar to the real samples. The dispersion of the GAN-generated samples is closer to the baseline than random, in all cases. The low dispersion of random samples is uncharacteristic of real samples, and might explain why their mean distance is lower than GAN samples in some cases.

In respect to the DualMyo data set, table 2 shows that GAN1 is significantly better than the random distribution in mean distance, for all classes. However, the standard deviation of the distance, which measures the dispersion within samples of the same class, is generally significantly lower than that of the data set. While the dispersion is comparatively low, it is significant, as seen in figure 4. It also indicates that the training of the GAN has not collapsed into generating a single type of sample.

Owing to the similarity between generated and trained samples, and the reasonable intra-class dispersion of the generated samples, we can conclude that the generators are successfully trained.

3.6 Discriminator Performance

The objective of the presented methodology is to improve the real-world performance of the discriminator. The performance is strongly tied to the accuracy of the classifier model, i.e., the number of successful classifications over the total number of gesture samples. However, the accuracy does not reflect the rate of non-gestures classified as gestures, which may happen in real-world conditions. The metric of interest for this type of problem is

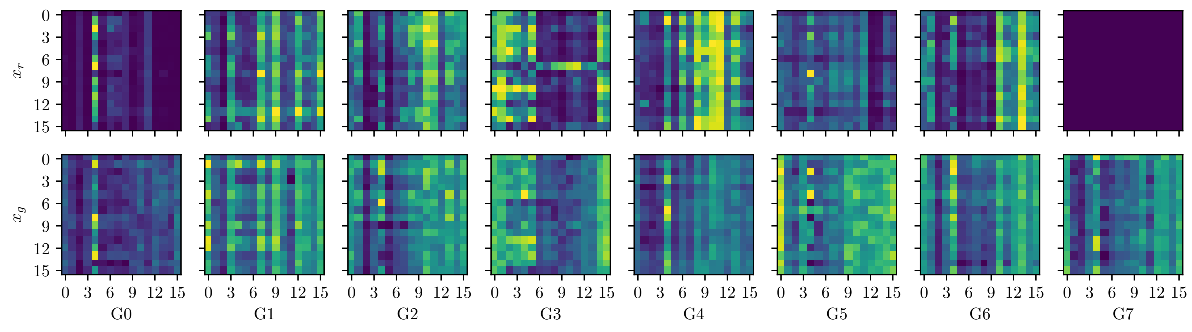

Figure 4: Comparison between real (x_r) and generated samples (x_g) of each class of the UC2018 DualMyo. G7 is a class created by the GAN, so it does not have a real class equivalent.

Table 1: Similarity performance indicators between samples of the UC2017 SG/DG data set, Gaussian noise samples and GAN-generated samples.

	Baseline		GAN		Random	
Class	Mean	Std	Mean	Std	Mean	Std
0	4.32	1.83	5.06	2.41	4.49	1.60
1	4.19	1.45	3.96	1.38	4.85	0.77
2	4.33	1.54	4.19	1.41	4.45	0.97
3	4.56	1.84	3.95	1.20	4.22	1.52
4	2.84	1.01	2.58	0.96	3.70	0.76
5	4.57	1.85	4.29	1.66	6.35	1.07
6	4.82	1.72	4.32	1.49	6.27	1.36

Table 2: Similarity performance indicators between samples of the UC2018 DualMyo data set, Gaussian noise samples and GAN-generated samples.

	Baseline		GAN		Random	
Class	Mean	Std	Mean	Std	Mean	Std
0	1.43	0.63	1.44	0.33	4.74	0.20
1	3.20	1.40	3.55	0.86	5.06	1.58
2	2.28	0.69	2.10	0.57	2.47	0.58
3	4.03	1.66	4.09	0.93	6.36	1.89
4	2.32	0.71	2.18	0.63	4.02	0.70
5	1.95	0.77	2.00	0.85	2.55	0.46
6	2.44	0.83	2.38	0.81	3.24	0.47

the prediction accuracy.

The analysis of the discriminator's performance is done as a two-step problem. The first step is the binary classification problem of novelty detection, i.e., determining whether a new sample belongs to one of the trained classes or not. The second problem is multi-class classification, where a prediction is performed to find what the class of a sample is, within the trained classes subset. The chosen performance metric is the classification accuracy (13). The novel detection accuracy (NDA) is defined as the fraction of novel samples that are correctly identified. On the other hand, gesture classification accuracy (GCA) is the fraction of correctly discriminated samples that belong to new classes.

$$\text{Accuracy} = \frac{\text{Number of correct predictions}}{\text{Total number of predictions}} \qquad (13)$$

The working hypothesis is that the data set augmentation with the previously described GANs improves the classification of gestures and unsupervised classification of novel gestures. The baseline performance is established by training a discriminator model with the data set samples. Following that, since we found that presence of stochastic labels on the data set helps training the GAN, we test their specific contribution without using the GAN framework. Finally, we test the performance of the discriminator trained within the GAN framework and retrained with real and generated samples:

Baseline: Discriminator trained regularly;

Test 1: Discriminator trained with stochastic target vectors;

Test 2: Trained GAN discriminator (online augmentation);

Test 3: Trained GAN discriminator retrained with real and generated samples (offline augmentation).

Additionally, these tests are repeated without the threshold defined in (11), and with the threshold tuned so that the GCA is around 95%, 90% or 85%. The tests were performed on the UC2017 SG and UC2018 DualMyo data sets, described in section 3.2.

The discriminator performance was measured in several settings. To ensure that the differences between them are due to the proposed methodology, the structure of the discriminator, figure 2, is fixed for all tests. Additionally, all networks are initialized with the same weights and the data set splits (training, validation and testing) are fixed as well.

The baseline test consists of the performance measurements calculated with the outputs of a discriminator that was trained as a regular neural network. The training hyperparameters were optimized for this purpose. The Adam optimizer was used with a learning rate of 0.01. The training process was halted using the early stopping technique, where the loss on the validation data is monitored online. The training process is stopped when the validation loss stops decreasing for 12 consecutive epochs in order to prevent over-fitting on the training data. Since this is a rather small training data set in a simple network, the fitting process takes just a few seconds.

In Test 1, the discriminator performance test is the same as the baseline, except that the training targets are stochastic vectors, as described in (6). The maximum value of the target vector of each sample was sampled before training from a uniform distribution $\mathcal{U}\sim(0.8, 1.0)$.

The second and third tests use the trained GAN with the setup described in section 3.4. For Test 2, the discriminator is used as trained in the GAN framework. For Test 3, it is retrained afterwards with stochastic labels and the training set is augmented offline with about +50% in number of samples.

All tests were repeated with different values of thresholds τ for the final classification decision, as defined in (11). The no-threshold case is equivalent

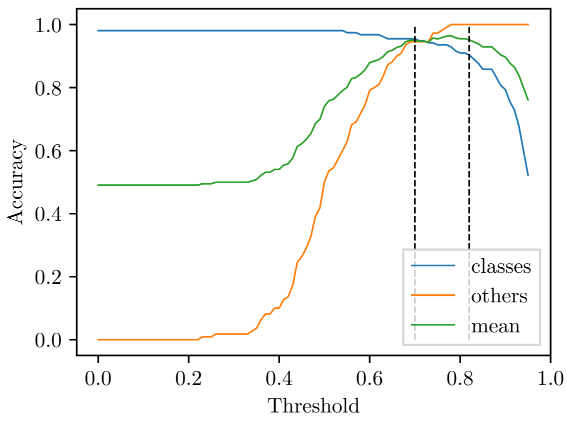

Figure 5: Trade-off between test split GCA and NDA a function of the decision threshold, on the UC2018 DualMyo data set. The two vertical lines correspond to the $p = 0.95$ and $p = 0.90$ thresholds.

to setting $\tau = 0$. An example of the threshold optimization process on the UC2018 DualMyo data set is shown on figure 5. This figure shows that when no decision threshold is set, the GCA is at its maximum while the NDA is at its minimum. As the threshold increases, the GCA (classes) decreases as expected, but the NDA (others) increases faster. Therefore, it is possible to numerically find an optimum balance between GCA and NDA. This balance was roughly set in the two data sets as the maximum GCA attained in the baseline test minus 5% and 10%. For example, if the maximum GCA is 100%, the threshold is tuned so that the GCA is at least 95% or 100%.

The results of all tests on the UC2017 SG data set are shown on table 3. When no decision threshold is set, the baseline GCA and NDA are 94.7% and 0.0%, respectively.

The baseline test results show that the discriminator's NDA increases from 0 to 37.6% while the GCA decreases by about 9% ($p = 0.85$). The use of stochastic labels on Test 1 show a significant improvement of NDA to 83.0% ($p = 0.90$, $\tau = 0.67$) and 91.0% ($p = 0.85$, $\tau = 0.81$). The augmented discriminator of the GAN, Test 2, yields a further improvement to 90.2% ($p = 0.90$, $\tau = 0.93$) and 92.0% ($p = 0.85$, $\tau = 0.95$). However, retraining the discriminator on Test 3 shows a decrease in performance, when compared to the discriminator

Table 3: Accuracy of D0's predictions on the test split. The *Class* and *Others* columns correspond to the trained and *others* classes, respectively.

	$\tau = 0$			Accuracy (%), $p = 0.90$				Accuracy (%), $p = 0.85$			
	Class	Others	Mean	Class*	Others	Mean	$\tau_{0.90}$	Class*	Others	Mean	$\tau_{0.85}$
Baseline	94.7	0.0	47.4	90.0	31.2	60.6	0.92	86.3	37.6	62.0	0.96
Test 1	95.0	0.0	47.5	90.0	83.0	86.5	0.67	85.0	91.0	88.0	0.81
Test 2	95.8	0.0	47.9	90.3	90.2	90.2	0.93	85.0	92.0	88.5	0.95
Test 3	95.3	0.0	47.6	90.0	83.2	86.6	0.66	85.5	90.0	87.8	0.77

*Closest accuracy value to p after threshold optimization.

Table 4: Accuracy of D1's predictions on the test split. The *Class* and *Others* columns correspond to the trained and *others* classes, respectively.

	$\tau = 0$			Accuracy (%), $p = 0.95$				Accuracy (%), $p = 0.90$			
	Class	Others	Mean	Class*	Others	Mean	$\tau_{0.95}$	Class*	Others	Mean	$\tau_{0.90}$
Baseline	98.1	0.0	57.4	96.1	17.3	63.4	0.95	96.1	17.3	63.4	0.95
Test 1	100.0	0.0	58.5	95.5	90.9	93.6	0.63	91.0	95.5	92.9	0.71
Test 2	98.1	0.0	57.4	95.5	94.5	95.1	0.70	90.3	100.0	94.3	0.82
Test 3	100.0	0.0	58.5	95.5	72.7	86.0	0.69	90.3	88.2	89.4	0.82

*Closest accuracy value to p after threshold optimization.

augmented online with GANs.

The discriminator tests were repeated for the UC2018 DualMyo data set and the results are shown in table 4. The behaviour on this data set is similar to that seen on the previous data set. The classification accuracy without a decision threshold ($\tau = 0$) is above 98.1% in this case, so the optimization targets for this threshold were set to 0.95 and 0.90. There is a significant increase in NDA between the baseline and the remaining tests. The NDA increases from 17.3% in the baseline to 94.5% on Test 2 ($p = 0.95$), or 100.0% when the classification decision threshold is optimized so that $p = 0.90$. However, similarly to the results on the previous data set, Test 3 shows a large drop in NDA to 72.7%.

The results show that in all data sets, the proposed methodology greatly increases NDA without significantly impacting GCA, as seen in tables 3 and 4. The baseline results show that the discriminator is not capable of detecting novelty in any circumstance. Additionally, the decision threshold does not change substantially after optimization in the baseline test (0.92 to 0.96). This result indicates that most of the predictions done by the neural network show high scores despite the sample's provenance. The use of stochastic target vectors in place of one-hot vectors lowers the prediction scores while maintaining the network's discriminability. Test 1 shows that it is possible to increase NDA while maintaining a high GCA with thresholds as low as 0.67 on the UC2017SG data set. The online data augmentation enabled by GANs yields a further improvement in NDA, as demonstrated for all data sets in Test 2. The offline data set augmentation tests (Test 3) show lower NDA than online augmentation. This is probably explained by the lower variance of the samples generated by a fully trained generative network (offline augmentation), when compared to the samples created by the dynamic generative model in the GAN framework (online augmentation).

4 Conclusion

In this work we implemented a novel way of solving the issue of classification of out-of-vocabulary gestures. Very often, these gestures are classified as an existing class and are difficult to remove with a classification threshold. The proposed solution has two components: (1) the use of a generative model (GAN) to augment the data set online with new generated samples, (2) the use of stochastic target vectors to decrease the average prediction score, thus facilitating threshold tuning. Finally, a threshold is set on the prediction score in order to make a final classification.

Tests were performed on two data sets to compare the discrimination capability of a neural network with and without the proposed changes. The results show that the use of stochastic target vectors improves significantly the novelty detection accuracy while maintaining a high classification accuracy. Furthermore, the online augmentation of the discriminator's training data set with a GAN yields a further improvement. However, while offline augmentation may offer better classification accuracy, it showed worse performance on novelty detection when compared to both online augmentation and stochastic target vectors. The results on the UC2017 SG data set showed a maximum classification accuracy of 95.8%. Without threshold tuning, the NDA is 0.0% in all cases. With threshold tuning, the NDA increased from 0.0% to 90.2% using the GAN framework coupled with stochastic target vectors, which is significantly better than the baseline (31.2%). The results on the UC2018 DualMyo data set showed a similar behaviour, where the proposed methodology had a NDA of 94.5% and the baseline 17.3%.

A major challenge of the proposed methodology is the successful training of the GAN. The generative model performance can still be improved in order to generate more diverse samples, even though the performance achieved in this work resulted in an improvement of NDA. Despite the success, further validation of the methodology should be done on richer data sets. Further work should also be done on the generation of time-series.

References

[1] Pedro Neto et al. "Real-time and continuous hand gesture spotting: An approach based on artificial neural networks". In: *2013 IEEE International Conference on Robotics and Automation* (2013), pp. 178–183. DOI: 10.1109/ICRA.2013.6630573.

[2] M. A. Simão, P. Neto, and O. Gibaru. "Unsupervised Gesture Segmentation by Motion Detection of a Real-Time Data Stream". In: *IEEE Transactions on Industrial Informatics* in press.99 (2016), pp. 1–1. ISSN: 1551-3203. DOI: 10.1109/TII.2016.2613683.

[3] Marco A.F. Pimentel et al. "A review of novelty detection". In: *Signal Processing* 99 (June 2014), pp. 215–249. ISSN: 0165-1684. DOI: 10.1016/J.SIGPRO.2013.12.026. URL: https://www.sciencedirect.com/science/article/pii/S016516841300515X.

[4] Xuemei Ding et al. "An experimental evaluation of novelty detection methods". In: *Neurocomputing* 135 (July 2014), pp. 313–327. ISSN: 0925-2312. DOI: 10.1016/J.NEUCOM.2013.12.002. URL: https://www.sciencedirect.com/science/article/pii/S0925231213011314.

[5] Diederik P. Kingma et al. *Semi-supervised Learning with Deep Generative Models.* 2014. URL: http://papers.nips.cc/paper/5352-semi-supervised-learning-with-deep-generative-models.

[6] Jost Tobias Springenberg. "Unsupervised and Semi-supervised Learning with Categorical Generative Adversarial Networks". In: (Nov. 2015). arXiv: 1511.06390. URL: http://arxiv.org/abs/1511.06390.

[7] Ian J Goodfellow et al. "Generative Adversarial Nets". In: (2014). URL: https://arxiv.org/pdf/1406.2661.pdf.

[8] Wei Han et al. "A semi-supervised generative framework with deep learning features for high-resolution remote sensing image scene classification". In: *ISPRS Journal of Photogrammetry and Remote Sensing* (Nov. 2017). ISSN: 0924-2716. DOI: 10.1016/J.ISPRSJPRS.2017.11.004. URL: https://www.sciencedirect.com/science/article/pii/S0924271617303428.

[9] Bin Huang et al. "High-quality face image generated with conditional boundary equilibrium generative adversarial networks". In: *Pattern Recognition Letters* 111 (Aug. 2018), pp. 72–79. ISSN: 0167-8655. DOI: 10.1016/J.PATREC.2018.04.028. URL: https://www.

sciencedirect.com/science/article/pii/S016786551830148X.

[10] Jie Li et al. "WaterGAN: Unsupervised Generative Network to Enable Real-time Color Correction of Monocular Underwater Images". In: *IEEE Robotics and Automation Letters* (2017), pp. 1–1. ISSN: 2377-3766. DOI: 10.1109/LRA.2017.2730363. URL: http://ieeexplore.ieee.org/document/7995024/.

[11] Xiaofeng Mao et al. "Semantic invariant cross-domain image generation with generative adversarial networks". In: *Neurocomputing* 293 (June 2018), pp. 55–63. ISSN: 0925-2312. DOI: 10.1016/J.NEUCOM.2018.02.092. URL: https://www.sciencedirect.com/science/article/pii/S0925231218302728.

[12] Yuan Yuan, Chunlin Tian, and Xiaoqiang Lu. "Auxiliary Loss Multimodal GRU Model in Audio-Visual Speech Recognition". In: *IEEE Access* 6 (2018), pp. 5573–5583. ISSN: 2169-3536. DOI: 10.1109/ACCESS.2018.2796118. URL: http://ieeexplore.ieee.org/document/8279447/.

[13] Yuki Saito, Shinnosuke Takamichi, and Hiroshi Saruwatari. "Statistical Parametric Speech Synthesis Incorporating Generative Adversarial Networks". In: *IEEE/ACM Transactions on Audio, Speech, and Language Processing* 26.1 (Jan. 2018), pp. 84–96. ISSN: 2329-9290. DOI: 10.1109/TASLP.2017.2761547. URL: http://ieeexplore.ieee.org/document/8063435/.

[14] X. Yuan et al. "Weighted Linear Dynamic System for Feature Representation and Soft Sensor Application in Nonlinear Dynamic Industrial Processes". In: *IEEE Transactions on Industrial Electronics* 65.2 (Feb. 2018), pp. 1508–1517. ISSN: 0278-0046. DOI: 10.1109/TIE.2017.2733443.

[15] Yang Li et al. "A Generative Model for category text generation". In: *Information Sciences* 450 (June 2018), pp. 301–315. ISSN: 0020-0255. DOI: 10.1016/J.INS.2018.03.050. URL: https://www.sciencedirect.com/science/article/pii/S0020025518302366.

[16] Augustus Odena, Christopher Olah, and Jonathon Shlens. "Conditional Image Synthesis With Auxiliary Classifier GANs". In: (Oct. 2016). arXiv: 1610.09585. URL: http://arxiv.org/abs/1610.09585.

[17] Jun-Yan Zhu et al. "Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks". In: (Mar. 2017). arXiv: 1703.10593. URL: http://arxiv.org/abs/1703.10593.

[18] Martin Arjovsky, Soumith Chintala, and Léon Bottou. "Wasserstein GAN". In: (Jan. 2017). arXiv: 1701.07875. URL: http://arxiv.org/abs/1701.07875.

[19] Miguel Simão, Pedro Neto, and Olivier Gibaru. *UC2017 Static and Dynamic Hand Gestures*. 2018. DOI: 10.5281/zenodo.1319659. URL: https://doi.org/10.5281/zenodo.1319659.

[20] Miguel Simão, Pedro Neto, and Olivier Gibaru. *UC2018 DualMyo Hand Gesture Dataset*. 2018. DOI: 10.5281/zenodo.1320922. URL: https://doi.org/10.5281/zenodo.1320922.

[21] Diederik P. Kingma and Jimmy Ba. "Adam: A Method for Stochastic Optimization". In: *CoRR* abs/1412.6980 (2014). arXiv: 1412.6980. URL: http://arxiv.org/abs/1412.6980.