# Introduction

In my research I have experimented with two different approaches for this classification problem:

1. Feature extraction with short time Fourier transform (STFT) and classification of the acquired spectrograms with a simple convolutional neural network (CNN). *# source code in filter.py and cnn\_filter.py*
2. Spike classification based on waveforms with K nearest neighbors (KNN).  *# source code in clusters.py*

Both of the approaches showed good performance on training data. The real challenge lies in separating neural signals from noise in any given experiment. Wrong implementation of filtering (band pass filter in this case) can dramatically change the spike shapes.

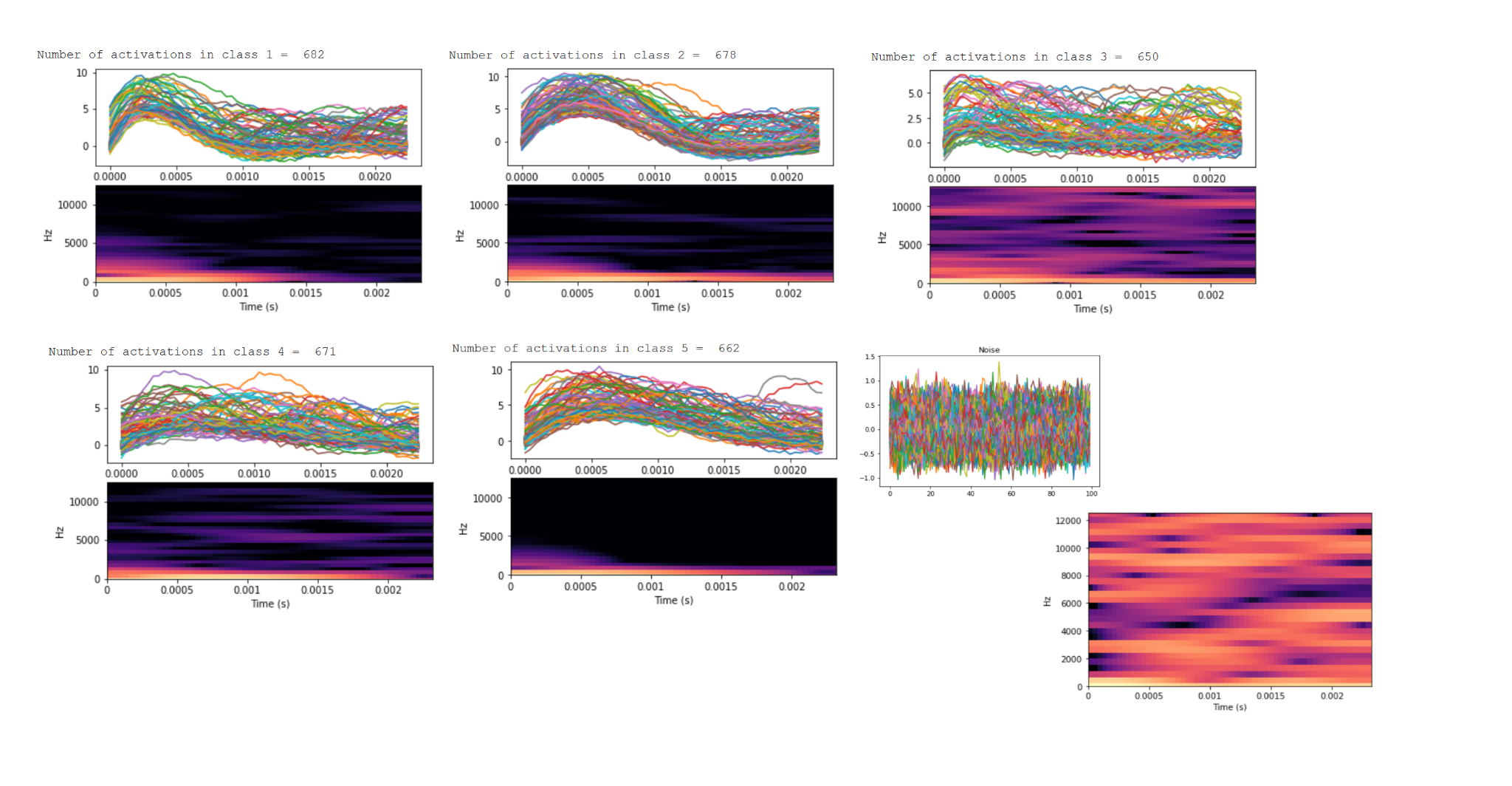
Therefore three different approaches of “filtering” were compared:

1. Creating an additional class of ‘noise’ and training CNN. *# source code in cnn\_filter.py*
2. Band pass filter function using butter with lfilter.
3. Band pass filter function using butter with filtfilt.

Spikes were detected using an amplitude threshold function.

# Performance metrics

Since classes in the training data are very balanced ((fig. 1) shows the number of activations in each class is approximately the same), both of the aforementioned approaches use **accuracy** as their performance metric.

In CNN approach accuracy is calculated at the end of each training epoch and for validation data using evaluation method for “Sequential” from keras.models.

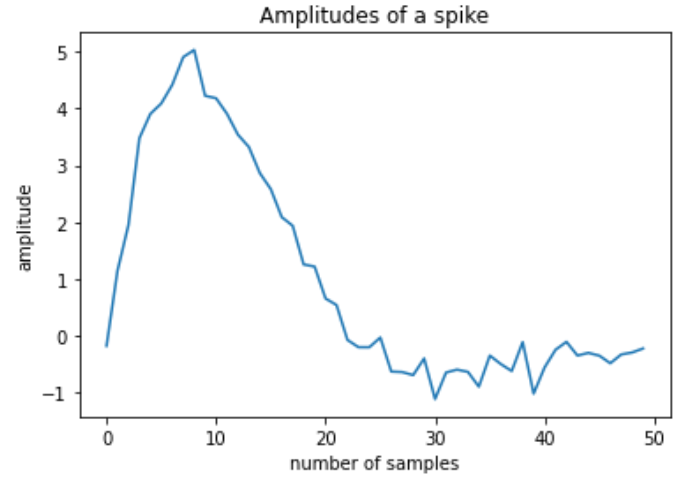
*fig.1*

*Number of samples, all waveforms, spectrogram of a random member in each class and noise in training data.*

In KNN it is done with sklearn.metrics.accuracy\_score.

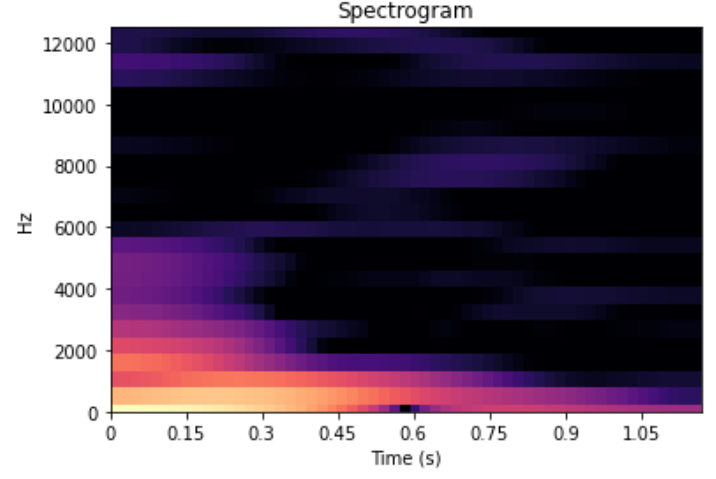
# First solution: CNN

## 1.Preparing the training data

The **Short-time Fourier transform** (**STFT**) is used to determine the sinusoidal frequency of local sections of the signal as it changes over time. In order to compute STFT of each spike (40 samples with 25kHz frequency) it is divided into shorter segments of equal length and then the Fourier transform is computed separately on each shorter segment to reveal time-dependent signal’s spectrum.

This approach extracts features of the spikes unseen in the initial amplitude sequences(fig.2). Even if the waveforms’ shapes generated by two neurons of one class are quite different, their frequency decomposition (fig.3) should be very similar and, most importantly, this method works well in the situations where the signals of several neurons are overlapping.

The algorithm for data preparing in for KNN:

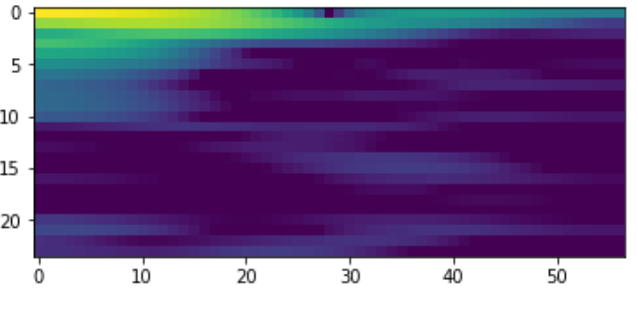
1. Take a spike from the data: 40 samples after index.
2. Slice the samples into intersecting frames. Add a padding frame (for convolution).
3. Perform STFT on each frame to get an array of frequencies.  *fig. 2*
4. Generate a spectrogram using librosa library.
5. Normalize spectrogram to an image pixel's values. 
6. Save the image.
7. Repeat for all the spikes listed in “Index”

Now we have a set of images stored as an array of numpy arrays. They are going to be split in between training (3426 images) and validation (570 images).

Since we work with keras, indexes (labels) are transformed to binary and now go from 0 to 4 (instead of from 1 to 5).

*fig. 3*

*Spectral decomposition of amplitudes in fig.2*

With this method an additional spike sorting algorithm was tested. As shown on fig.1 an additional class of noise was created (taken from the samples between the spikes, where the gap is long enough). It works well (up to 95% of accuracy), but obviously won’t be of much help to data from the experiment, since the noise is unique to the circumstances. It can be implemented if one has a recording of noise from an experiment separately, as well as labeled examples of spikes’ waveforms. 

*fig.4*

*Image of the spike from fig.3 and 2*

## 2. Building and compiling of the model

Once the data is ready, we need to define the architecture of the model and compile it with necessaryoptimizer function, loss function and performance metrics. The chosen architecture consists of 2 convolution layers followed by a pooling layer, a fully connected layer and softmax layer respectively. Multiple filters are used at each convolution layer, for different types of feature extraction. After both maxpooling and fully connected layers, dropout is introduced as regularization in our model to reduce overfitting problem.

I am using “categorical\_crossentropy” loss function as it is a multi-class classification problem. Since all the labels carry similar weight accuracy is the performance metric (as was mentioned above). A gradient descent technique called AdaDelta (adaptive learning rate) is used for optimization of the model parameters.

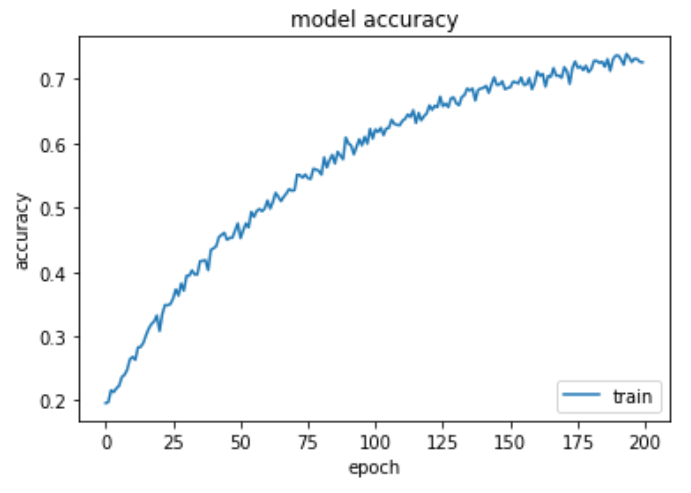
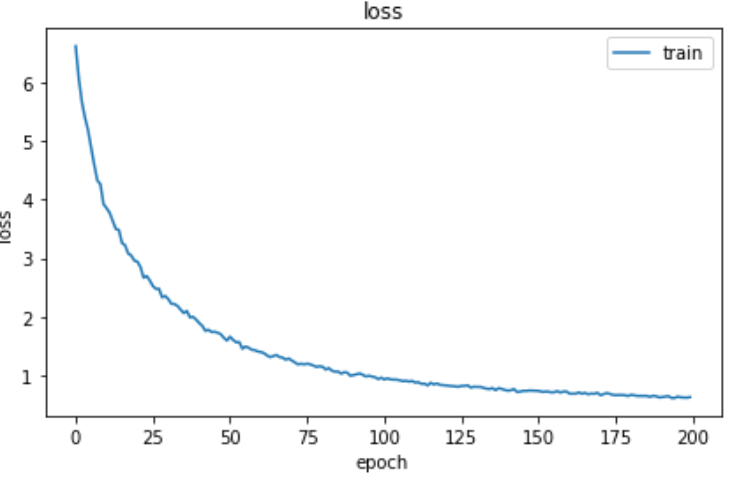
## 3.Training and evaluating the model

One epoch means one forward and one backward pass of all the training samples. So the batch size implies the number of training samples in one forward/backward pass.

Loss function and accuracy are calculated at the end of each epoch.

*fig.4*

*Accuracy and loss over epochs for model with 5 classes + noise class*



*fig.5*

*Loss and accuracy over epochs for spike classification (without noise)*

The implemented CNN gives accuracy of 85-90% on training and validation sets in the case of a simple spike classification.

When the class of noise is added the final accuracy is 95% .

So, this CNN is not too bad for the task, yet, since the spikes contain only 30-40 samples, the images are quite small, there are not too many features to extract from them. In addition, some information is lost when one gets rid of complex parts of Fourier transforms.

Since the accuracy for the noise sorting is much better, I believe this method can find its application in sorting spikes from the noise.

# Second solution: KNN

## 1.Visualise waveforms

Before attempting to clusterize waveforms, let’s see what we are dealing with:

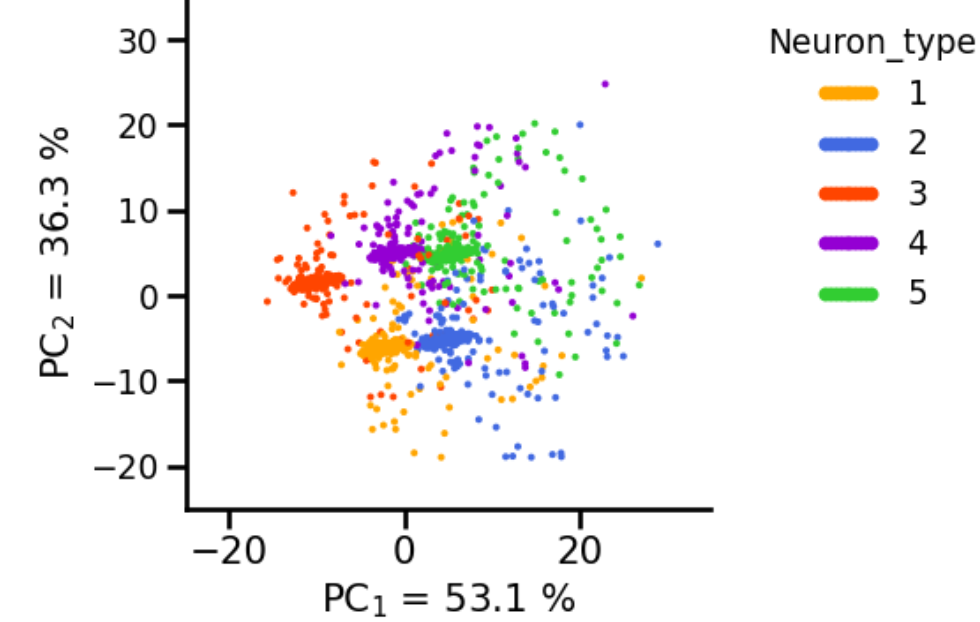
Every waveform contains 40 samples (or vector of features). We want to reduce the number of measurements to a minimum necessary to perform a further classification (ideally to keep 95% of the variance(fig. 7)).

## 2.Principal Components Analysis

So, in order to apply KNN, dimensionality needs to be reduced.

The principal components of a collection of points are a sequence of vectors, where the 𝑖-th vector is the direction of a line that best fits the data while being orthogonal to the first 𝑖-𝟣 vectors. A best-fitting line is defined as one that minimizes the average squared distance from the points to the line.

With them one can change the basis of the data, using only the first few principal components and ignoring the rest to reduce dimensionality.

Plotting the first two projections: 

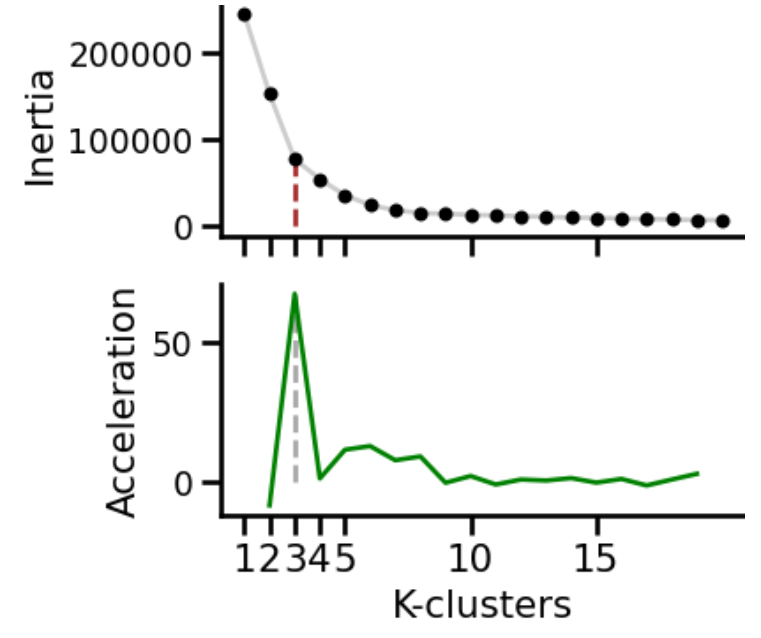
*fig. 7*

*Amount of PC for proper variance fig. 8*

*Clusters with first 2 components*

## 3. Clustering

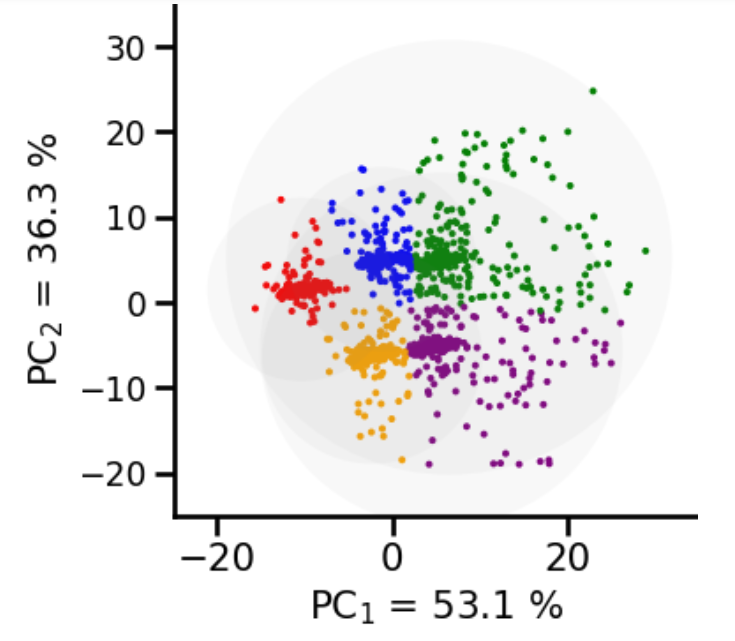
To determine whether we can form our spikes in 5 clusters, we need to compute the cluster after which the distortion/inertia start decreasing linearly. The distortion or inertia of a clustering result is the sum of squared differences between an observation and its corresponding centroid.

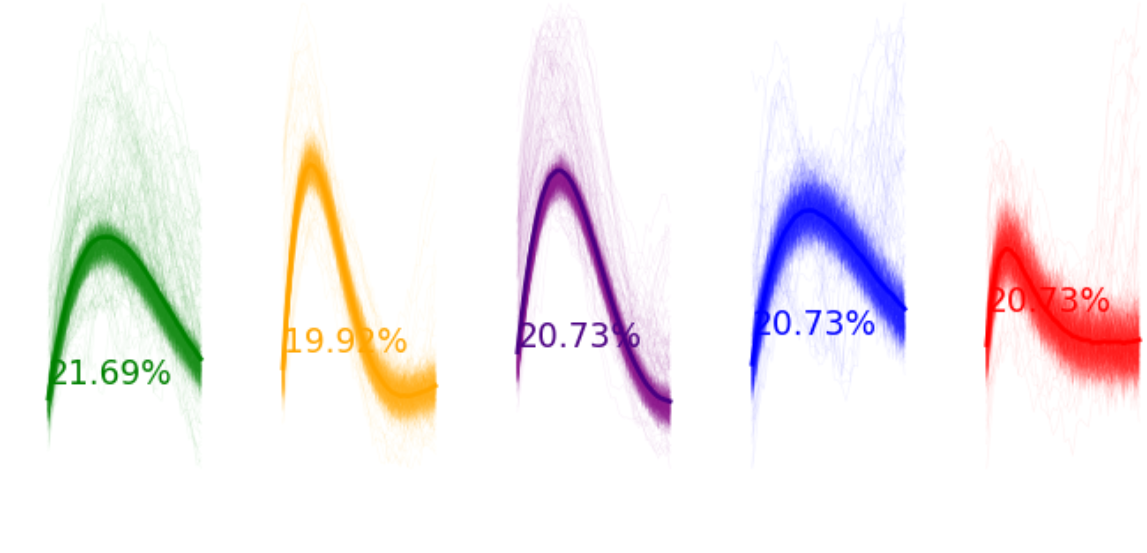
*fig.9* 

Inertia is the sum of squared errors. Thus, the inertia is the sum of squared euclidean distances for each point to its closed centroid:

, where is the number of points, is the k-centroid, and - euclidean distance.

To detect the minimal number of clusters, we can compute the second derivative of the inertia.

As shown in fig. 9, after 5 clusters inertia is still a nonlinear function. So, we can apply KNN without worrying too much. 

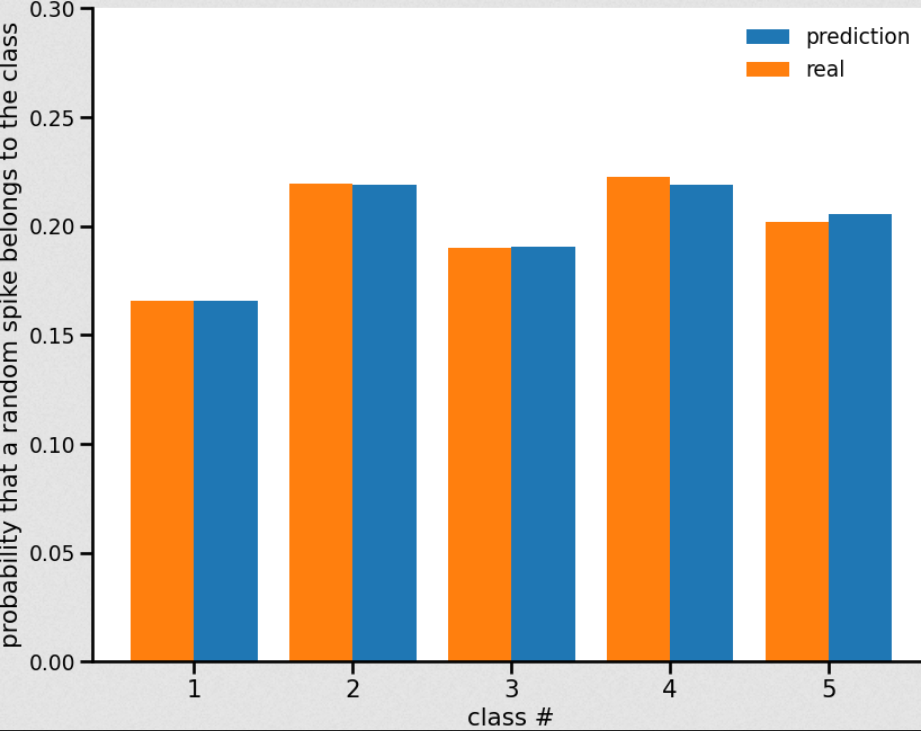


## 

*fig. 10*

*Clusters and their waveforms*

Fig. 10 shows the percentage of waveforms in each cluster. When we are sure that spike clusters are quite well-separable, now it’s time for KNN.



## 4. KNN model

Formally, the method is based on the compactness hypothesis: if the metric of the “distance” between objects (spikes) is introduced successfully, then same-class spikes are much more often found in the same class than in different ones (fig. 11).

To classify each of the spikes, one has to perform the following operations:

-Calculate the “distance” to each of the training sample spikes.

-Select the spikes of the training sample, the “distance” to which is minimal. *fig. 11*

-The class of the spike is the one most commonly found among closest neighbors.

The final (after optimization) accuracy of KNN on the validation set is 0.9701046337817638 or 97%. Which is substantially better than in CNN approach for a simple (without noise class) classification. Since the spikes form solid clusters it is better to use a bruteforce KNN method without losing information in STFT.

# Optimization

## 1.KNN optimization

The quality of the classification/regression by the KNN method depends on several parameters:

* number of neighbors
* metric of distance between objects
* weights of neighbors (the “further” is the neighbor, the smaller is its coefficient)

In order to find the best parameters one needs to create and train a model with different ones and compare the results.

Let's select the parameters of our model using GridSearch, implementing it through the grid from 1 to 12 of , trying different metrics:, ,,

and weighting strategies: “uniform” (all points in each neighborhood are weighted equally) and “distance” (weight points by the inverse of their distance. in this case, closer neighbors of a query point will have a greater influence than neighbors which are further away).

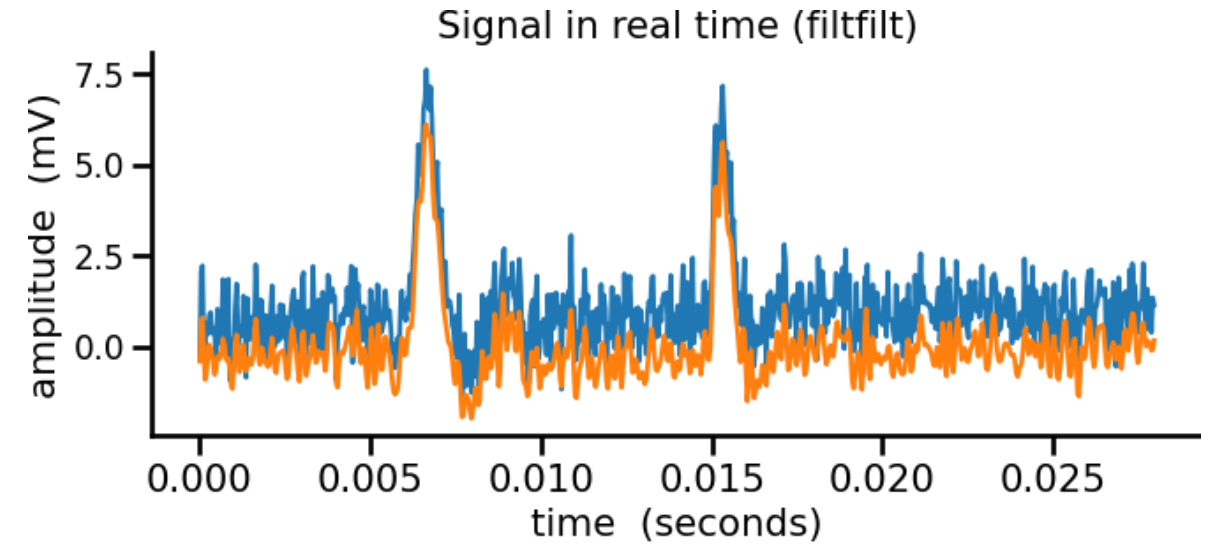
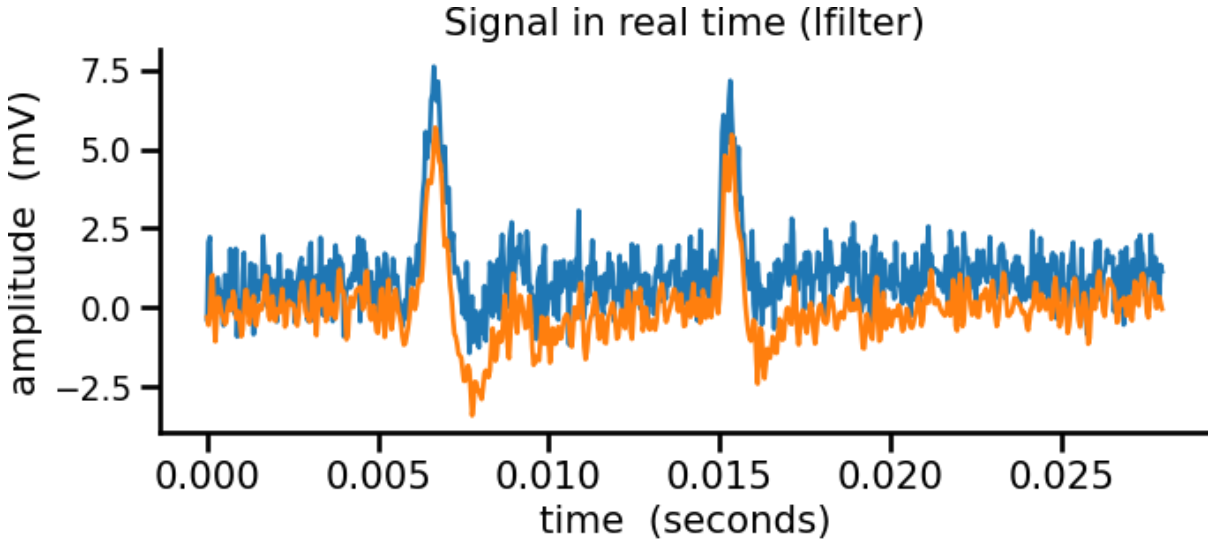
Best parameters found: {'metric': 'manhattan', 'n\_neighbors': 3, 'weights': 'distance'}

Accuracy after optimization: 97%

## 2. Optimizing filters for the experimental data

* *scipy.signal.lfilter* filters data along one-dimension with an IIR or FIR filter.
* *scipy.signal.filtfilt* applies a digital filter forward and backward to a signal. This function applies a linear digital filter twice, once forward and once backwards. The combined filter has zero phase and a filter order twice that of the original.

Let’s see how both of them transform the experimental data with low-pass frequency=30Hz and high=6000Hz (the frequencies were estimated from signal spectrograms (fig. 1)) :



*fig. 12*

*Blue: before*

*Orange: after filter*

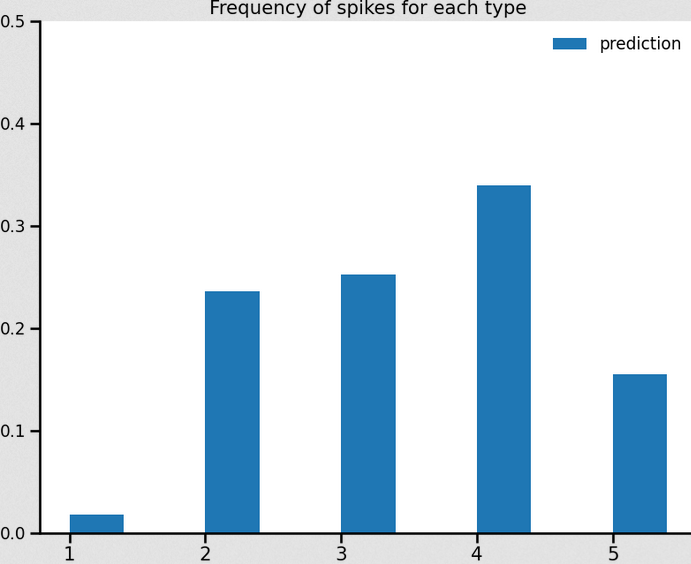
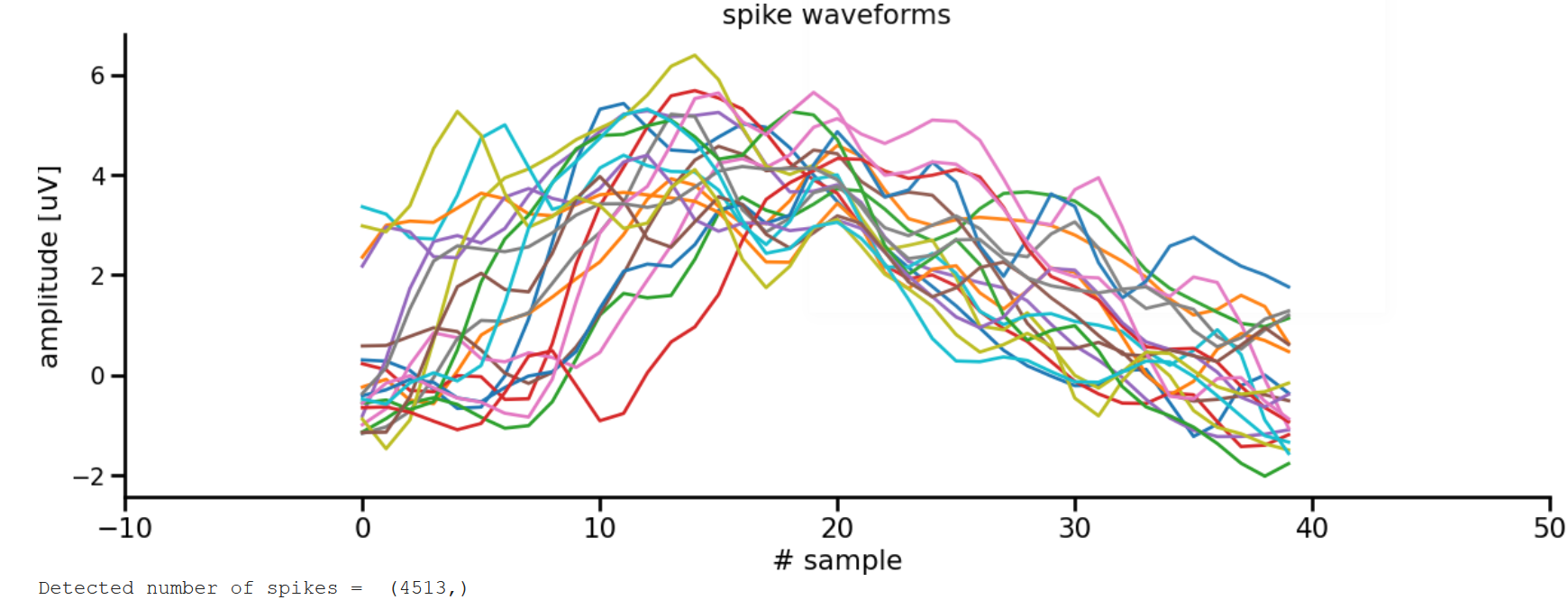
As one can observe (fig. 12), filtfilt doesn’t distort the spikes as much as lfilter. Causal filters (any recursive or IIR filter) can introduce phase distortions which seriously change the spike shapes. So, in a way it’s the same problem as in Fourier transform: information about phase is lost.

So, filtfilt is the choice.

## 3. Spike detection optimization

From filtered data spikes are detected using an amplitude threshold. If a high threshold is used missing spikes occur. A low threshold will get us lots of false positives due to noise crossing a low threshold.

Using the estimation based on the median diminishes the interference of the spikes.



*fig. 14*

*Predictions for the experiment*

*fig. 13*

*Selected experimental spikes after filtering*

# Conclusion and confidence level

With adequate filtering, simple methods such as KNN work quite well. In a given case, about 4000 spikes were sorted and classified in between neuron types as shown on fig. 14. The result seems reasonable.

The first proposed solution requires better filtered data, yet shows promising results on the training set.

An alternative solution of filtering was proposed with the first solution.

*Thr*=5

*σ*

*n*

*σ*

*n*

=*median*{

|*x*|

0.6745

}