Pázmány Péter Catholic University
Faculty of Information Technology and Bionics

Multi-domain classification of cardiac signals with Deep Neural Networks

Botos Csaba & Hakkel Tamás

2017

A review submitted in partial satisfaction of the requirements of
Council of Scientific Students' Associations

Supervisors: PhD. András Horváth, PhD. István Z. Reguly, Márton Áron Goda

Abstract

Recent success of convolutional neural networks in computer vision have motivated applying the same representation learning methods to signal processing as well. The novel data driven approaches can make good use of a labeled dataset, lacking well-defined mapping between the input-output pairs. Our main contribution is a special design of preprocessing steps that boost the generalization, and stabilize the training of the underlying classifier algorithm. We propose a deep neural architecture for real-time automated suggestions of possible cardiac failures that is able to detect class invariant anomalies, using a single channel of a portable ECG device. The uniqueness of our approach is that we deploy two parallel networks learning temporal and spectral domain representations from scratch, trained side-by-side. We are aware of the possible flaws in the ground truth labels, so our method offers visual reasons to confirm its predictions. Applying visualization methods for extracting self-learned representations we are able to draw the attention of the medical staff to the main contributing signal patterns that play the most important roles in the suggested failure.

Keywords: ***deep learning, residual network, fully convolutional network, time series, signal processing, ECG, atrial fibrillation, af detection***

Acknowledgement

We would like to say thanks to our supervisors for their patience, guidance, and constant help throughout the last year: PhD Horváth András, PhD Reguly Z. István, Goda Máron Áron.

Also we must thank Dr. Herczeg Szilvia and Dr. Osztheimer István, who generously offered their help and time to teach us the basics of cardiology, supervise our program from the medical aspect, and give us useful advises.

Contents

1. Introduction

As soon as you have good mechanical technology, you can make things like backhoes that can dig holes in the road. But of course a backhoe can knock your head off. But you don't want to not develop a backhoe because it can knock your head off, that would be regarded as silly.

— Geoffrey Hinton

Some *believe* that once the computers will be better than us, they surely will have to wash off humans from the face of the earth, *fear* that AI brings nothing but more unemployment and darkness to the everyday life. On the contrary some may call these years as the beginning of the 4^{th} industrial revolution, *think* of Artificial Intelligence as the modern equivalent of electricity, that will extend our psychical capabilities, as once electricity extended physical limits, and *see* Neural Networks as a possible architecture to bring this bright future. In the meantime, it is still our responsibility to take the opportunity and make good use of these brilliant instruments. As bionic engineers, it fits perfectly our role to understand and model the fine-tuned mechanisms of adapting biological systems in nature. The main purpose of this work is two-fold: first we would like to improve our understanding of cardiac signals by revealing invariant patterns from raw ECG recordings using deep convolutional networks. Second, we believe these experiments help us to generalize our current practical knowledge about representation learning algorithms.

Neural Networks

Artificial Neural Networks, a branch of Machine Learning have been around for several decades, however the current wave, the deep learning era is the most recognizable one that drew attention of professionals from different fields and connected them. The variety of the applications of deep neural networks (DNN) is wide, since they are universal approximators in theory, which means if enough training samples, computation time and capacity is provided, any function can be reconstructed and modeled. With the theory aside, the solution space of applied Neural Network has its own practical boundaries.

Capabilities and limitations. Deep learning possibly brings the future of programming, where we no longer have to carefully design a set of rules for each different objective - whether we want to solve everyday tasks (e.g. describing an image in a few sentence) or highly domain specific problems (e.g. predicting the output of a numerically intractable chemical measurement). However there are still some major issues with building universal approximators: having insufficient training samples, reducing the computational complexity or transferring knowledge from a trained model on a biased distribution reveals that current architectures are relying strongly on domain-specific features, and do not generalize well. Also there are two major trends in DNN solutions - offline algorithms have no constraints on the evaluation time or power efficiency, rather focus on outputting predictions with the highest accuracy. On the contrary online algorithms sacrifice precision in order to evaluate test samples in real time by meeting complexity and capacity constraints.

Recent successes in computer vision. Most of recent breakthroughs occurred in the field of image processing, where huge amounts of unstructured data were already a great challenge for traditional algorithmic approaches. A few decades ago there were no signs that in the near future there would be computers able to tell our emotions in real-time from simple gray-scale images, automatically describe a scene for a visually impaired person, tell if someone is present on an image, show their faces from unseen angles, render them in 3D space, or even make them talk to us. However these are now publicly available algorithms, provided with training sets, and trained baseline models, and most importantly well documented experiments and evaluations. This enables everyone who has access to the internet to use, review, improve and share new ideas which led to exponential growth in interest of deep learning pulling both academic and industrial investments.

Graphical Processing Units The whole act was made possible by the growth of computational capacity of parallel processors, mainly motivated by the demands of the video-game industry - they developed in a complimentary way, hand in hand. Soon the number of cores in GPUs jumped to a level where easily distributed tasks could outperform complex sequential computations both in process-time and precision. In the past years development was driven mostly by AI. Since evaluating a single neuron's activity only requires the values of its direct inputs and can be implemented as a vector product embedded into a non-linearity (that will be later described), huge numbers of neurons can be evaluated at once making them a perfect match for the capabilities of GPUs.

Practice is the best of all instructors If someone wants to keep pace with the inventors and take part in the 4^{th} industrial revolution, one can find themselves in times of trouble. Not because of the lack of self-study materials, but the amount of available content. At the time of writing dozens open-source projects, and numerous free courses in deep learning are available that fit the personal level of understanding and style preference - still understanding these algorithms is not equivalent with being able to apply them. In order to strengthen our perception of the field, furthermore to reveal the potential of neural networks for larger audiences we entered a signal processing challenge. We have developed a model to improve medical attendance by providing automatic suggestions for cardiologists based on real-life recordings of portable AliveCor ECG devices.

Application for better purposes

Social relevance. Thousands of heart failures could be prevented with proper treatment if early signs were diagnosed in time. Over the years medical equipment advanced by involving some sort of artificial intelligence. We don't have to go far, probably every gas station in the area has a semi-automatic defibrillator, that has a sensor built in which recognizes whether the patient needs to be shocked or not — to prevent unnecessary reanimation. The basic idea is to relieve overwhelmed doctors by recognizing invariant patterns in the specific pathological situations. These patterns are taught at medical universities, and fine-tuned during years of practice — but turns out that volunteering cardiologists can submit their knowledge base to engineers who in return will automatize at least the trivial process to support the better treatment. In order to assist in the diagnosis, and make predictions based on previous cases we utilize Machine Learning techniques to combine professional knowledge and neural networks to achieve the lowest error rate on a classifying task. To emphasize the importance, and social relevance of our work we could add that the the number of deaths from AF increased to 193,300 deaths in 2015, from 29,000 in 1990 [20].

The real challenge. Atrial Fibrillation (AF) is the most common type of cardiac arrhythmia, still the state-of-the-art algorithm is under-performing on real data. This problem was the main motive behind the AF Challenge:

> The 2017 PhysioNet/CinC Challenge aims to encourage the development of algorithms to classify, from a single short ECG lead recording (between 30 s and 60 s in length), whether the recording shows normal sinus rhythm, atrial fibrillation (AF), an alternative rhythm, or is too noisy to be classified. There are various types of cardiac arrhythmias that may be classified by
>
> - Origin: atrial arrhythmia, junctional arrhythmia, or ventricular arrhythmia.
> - Rate: tachycardia ($>$ 100 beats per minute (bpm) in adults) or bradycardia ($<$ 60 bpm in adults).
> - Mechanism: automaticity, re-entry, triggered.
> - Atrio-ventricular Conduction: normal, delayed, blocked.

- Duration: non-sustained (less than 30 s) or sustained (30 s or longer).

> Previous studies concerning AF classification are generally limited in applicability because 1) only classification of normal and AF rhythms were performed, 2) good performance was only shown on carefully-selected often clean data, 3) a separate out of sample test dataset was not used, or 4) only a small number of patients were used. It is challenging to reliably detect AF from a single short lead of ECG, and the broad taxonomy of rhythms makes this particularly difficult. In particular, many non-AF rhythms exhibit irregular RR intervals that may be similar to AF. In this Challenge, we treat all non-AF abnormal rhythms as a single class and require the Challenge entrant to classify the rhythms as 1) Normal sinus rhythm, 2) AF, 3) Other rhythm, or 4) Too noisy to classify.

———— Introduction to *Computing in Cardiology Challenge of 2017*[2]

The rules. A moderate set of single lead ECG samples (8528) recorded with a portable device called Kardia Mobile and labeled by a team of cardiologists is provided with the following classes: *noisy, normal, atrium fibrillation, other* or in their short form: $\sim, N, A, O$. The task is to develop a method that is able to classify a set of unseen (3658) prerecorded sample with the highest accuracy. At first sight the initial conditions are very encouraging, but soon after the first check of the training files it turned out that only a limited number of samples are given, exactly 8528 samples, and the data set is extremely imbalanced. Also as it was revealed during the last weeks, the ground truth labels had several errors, misleading the training process. We had several discussions in the early phase of development with professionals about possible classification errors, that even were trivial to us. Many suggestions were committed by the competitors, and as a result the organizers released an updated label reference.

Measuring device. As mentioned earlier, the data set is recorded by a device of the company AliveCor that is named Kardia Mobile (see figure 1.1). It is a relatively new, low-cost device that makes ECG recording surprisingly easy: The device has two small metal plates, the patient puts one or two of his or her fingers from one hand to one of the plates, and other fingers to the another plate, and when it is done, the device immediately starts to record ECG. The device has no displays, so it send the data a mobile phone, where an app receives and processes the input.

Problems. While it might sound very appealing to have a small, simple, and low-budget device that the patient can bring anywhere, it has some problematic drawbacks as well. First, the connection between the fingers and the plates are not perfect, thus significant amount of noise is introduced to the recordings – even a very little movement of the patient can produce detectable noise. Second, due to the simple design of the design, it records only single lead ECG, meaning the device watches only one channel

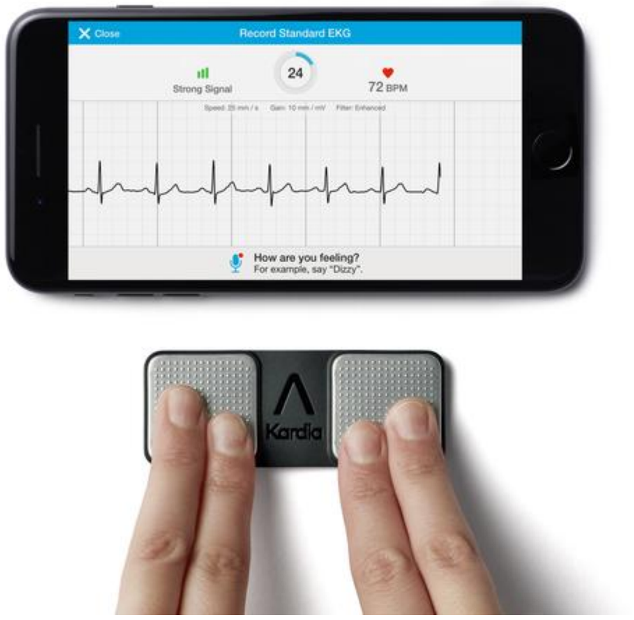


Figure 1.1: Photo of the measuring device of the company AliveCor called Kardia Mobile. Data set of the challenge is recorded by devices of that type. Image Credits: AliveCor

Type	# recording	Time length (s)				
		Mean	SD	Max	Median	Min
Normal	5154	31.9	10.0	61.0	30	9.0
AF	771	31.6	12.5	60	30	10.0
Other rhythm	2557	34.1	11.8	60.9	30	9.1
Noisy	46	27.1	9.0	60	30	10.2
Total	8528	32.5	10.9	61.0	30	9.0

Table 1.1: Length of recordings in training data set

that makes diagnosis much harder compared to the conventional 12-lead devices used in clinical practice. Third, the device is designed to record short recordings ranging from 9 to 61 seconds (see table 1.1). Whereas it is convenient to the patient, doctors generally records much longer recordings because atrial fibrillation is episodic, that is, it occurs rarely, so the shorter recordings is the lower is the likelihood that it catches an event.

Augmentation. Being aware of the problems, we inspected our possibilities to augment the data set because the key of successful training of neural network is the large, heterogeneous and balanced training set. We considered generating new recordings from the given data set by cropping, stretching the samples or even by adding some noise to them. Then we focused on the noisy class because that class had the least samples. We were thinking about to generate noisy samples from random noise, and we even asked cardiologists to produce some "blank" recording for us (i.e., the recordings device is not attached to a patient, but it recorded only noise. Besides, we made some experiments on creating recordings by a neural network. But finally we rejected all of these ideas because they did not we could not ensure that the new samples does not mislead our neural network by adding new (and possibly medically not correct) information to the training set. Lastly, we managed to get a device, and we planned to record some new samples, but it was not designed to help that kind of attempts (it transmits the data over ultrasound, and the app can save the data only as plots in pdf files.

Contributions and Outline

The outline of this work follows the same path we took to enter and complete the CinC Challenge of 2017.

Chapter 2 points out preliminary studies in signal processing, traditional machine learning algorithms and novel deep learning architectures - strong baselines for our initial experiments.

Chapter 3 walks trough the steps of understanding the relevant mathematical background we are working with: i.e. the core building blocks of *fully convolutional network* (FCN) and *recurrent neural networks* (RNN) and the most successful assemblies of them. In the appendix we also provide the analytical apparatus of supervised training the predecessor of FCNs and RNNs the *multi-layer perceptrons* (MLP), the back-propagation algorithm, and the advanced weight update policies.

Chapter 4 begins with defining the training and testing environment, used to develop and evaluate our approaches. Later in this chapter we introduce model modifications specifically designed for ECG classification in a logical order, describing our experiences for each major improvements applied and the motivations for the succeeding experiments.

Chapter 5 examines the medical relevance of the classifier: whether the model learns features that are specific indeed for the human heart or rather watches features that characterizes only the provided data set. First, it presents our methods to gain information about the quality of data set. Second, it shows the way how we inspected the features the model is looking for. Third, it describes out visualization technique that highlights the most relevant sections of the recordings.

Chapter 6 summarizes our contributions to the field of *Computing in Cardiology* and in general to signal processing with deep neural networks. During the final supervision of this work (2017 Q4) the results of CinC Challenge of 2017 have been published and presented on the annual conference in Rennes. In this chapter we will make conclusion on competition's results as well. Sharing ideas with the authors of the best performing solutions inspired further improvements on our current approach and opened opportunities for collaboration in the future - finally we will elaborate on both short and long-term plans.

2. Related Works

Our work is relying mainly on two disciplines: machine learning and medical signal processing. In this chapter we cover the details of previous attempts to classify electric signals produced by the cardiac conducting system. See Appendix A for a thorough description on how these patterns are formed.

Medical signal processing

Biological systems emit electromagnetic signals which can be measured with delicate sensors. Since the heart muscular fibers produce well defined rhythms of electrical waves during alternating phases of contraction and relaxation we can associate pattern anomalies detected with well-known cardiac disorders - such as AF.

Traditional approaches on AF detection. Previous approaches were relying on human-designed rules and decision trees. According to authors of one of the top algorithms in the challenge, it turns out that hundreds of possible rules may apply in a given case and another hundred may result in *false* predictions.

Still the most relevant features are the ones that are taught in med schools, for a comprehensive list of reliable features recognized by cardiologists see [63]. An excellent collection of preliminary studies concerning automated detection of AF was provided by the Challenge, which mainly lists the state of the art algorithms at the beginning of the competition:

> AF detectors can be thought of belonging to one of two categories: atrial activity analysis-based or ventricular response analysis-based methods. Atrial activity analysis-based AF detectors are based on the analysis of the absence of P waves or the presence of fibrillatory f waves in the TQ interval. Published methods to do this include: an echo state neural network [59], P-wave absence (PWA) based detection [38], analysis of the average number of f waves [14], P-wave-based insertable cardiac monitor application [61], wavelet entropy [3, 64] and wavelet energy [17]. Atrial activity analysis-based AF detectors can achieve high accuracy if the recorded ECG signals have little noise contamination and high resolution, but tend to suffer disproportionately from noise contamination [11]. In contrast, ventricular response analysis is

> based on the predictability of the inter-beat timing ('RR intervals') of the QRS complexes in the ECG. RR intervals are derived from the most obvious large amplitude feature in the ECG, the R-peak, the detection of which can be far more noise resistant. This approach may therefore be more suitable for automatic, real-time AF detection [9]. Published methods include: Poincaré plot analysis [4], Lorenz plot analysis [68], analysis of cumulative distribution functions [72], thresholding on the median absolute deviation (MAD) of RR intervals [44], histogram of the first difference of RR intervals [26], minimum of the corrected conditional entropy of RR interval sequence [47, 48], 8-beat sliding window RR interval irregularity detector [58], symbolic dynamics and Shannon entropy [5], sample entropy of RR intervals [56, 39, 12].
>
> It is worth noting that AF detectors that combine both atrial activity and ventricular response could provide an enhanced performance by combining independent data from each part of the cardiac cycle. Such detection approaches have included: RR interval Markov modeling combined with PR interval variability and a P wave morphology similarity measure [6] and a fuzzy logic classification method which uses the combination of RR interval irregularity, P-wave absence, f-wave presence, and noise level [60]. It is also worth noting that multivariate approaches based on machine learning that combines several of the above single features can also provide enhanced AF detection [57, 1, 43].

——— Introduction to *Computing in Cardiology Challenge of 2017*

Machine Learning

At the time of writing, supervised deep learning is a well-known data driven approach that is generally applicable to various fields of signal processing, however new inventions appear mostly on image processing and computer vision, largely thanks to the publicly available large-scale image database ImageNet [13]. A recent breakthrough was achieved by an architectural innovation, namely Residual Networks [21] that allow us to stack multiple layers in blocks whose output is added elementwise to its input, therefore these blocks will 1) only have to learn residual modifications in representation space 2) stabilize the gradient flow during backward propagation. Using residual blocks we can deploy networks with more parameters, enabling our parametrized function to approximate more complex mappings. Studies of He et. al [22] showed that a specific internal structure of the residual blocks and their twin branch in the computation graph, the skip-connections, are playing an important role in the trained networks' performance. Previous works applying Deep Learning for time series classifications helped us with the first steps. We based our initial experiments on strong baseline time-series classifiers trained from scratch [79], where the authors provided 3 candidate architectures that we also evaluated on the AF Challenge dataset. We soon learned that the receptive field of the classifier neurons in the last layer were too small to detect arrhytmias spanning multiple heartbeats. Knowing that some form of cardiac failures may also appear episodically in our training samples we considered the following advanced techniques: Dilated convolutions [82] and Long short-term memory (LSTM) networks [54] and their simplified architectures Gated recurrent units (GRU) described in [52]. Technically speaking RNNs provide a better solution for handling different length samples as well. We separately trained dilated convolutinal networks, and multi-layer recurrent networks in the same setting. The results of [8] motivated us to apply also end-to-end training of the convulutional feature extractors before the recurrent units, however the stacked architecture was unstable to be trained from scratch. During the challenge an alternative solution have been published [62], featuring a residual neural network which can exceed the precision and recall of individual experts in categorizing single-lead ECG recordings in 10 cardiac failure classes. The resemblance between their candidate network and ours confirmed the applicability of deep learning for single-lead ECG signals and suggested further investigation into our data augmentation and preprocessing steps. Restricted number of samples and class unbalance are well known issues of the AF Challenge, which motivated us to incorporate a technique listed in [7], namely random oversampling (ROS).

3. Deep Learning models

No one knows what the right algorithm is, but it gives us hope that if we can discover some crude approximation of whatever this algorithm is and implement it on a computer, that can help us make a lot of progress.

— Andrew Ng

We began the challenge with the hope that our previous experience on image processing and machine learning will pay off. A preliminary study of training neural networks with a thorough mathematical walk-trough is provided and can be found at Appendix C. Taking a closer look at the data set revealed three major obstacles for training deep neural networks. History of the label distribution can be seen on Figure 3.1.

Size. 8528 samples were provided for training and evaluating our algorithms and 3658 ECG samples were held back as a hidden test set until the announcement of the final results. Compared to lightweight benchmark datasets like CIFAR-10 [36] or MNIST [41] where 10000 samples are provided simply for evaluation purposes, and the training set is almost tenfold of the challenge's.

Balance. The objective to classify a sample into 4 different classes can mislead those who are not aware of the class imbalance of the AF training set and the advanced evaluation metric. By first assumption we could say that the worst performance is 25% because random guess would yield $1/N$ score for N classes. For this particular reason the authors of the challenge set the metric to the average of the F1 scores, which is basically a generalized score of the *precision* and *recall* of the underlying algorithm:

$$\text{F-score} = 2 \times \frac{\text{precision} \times \text{recall}}{\text{precision} + \text{recall}}$$

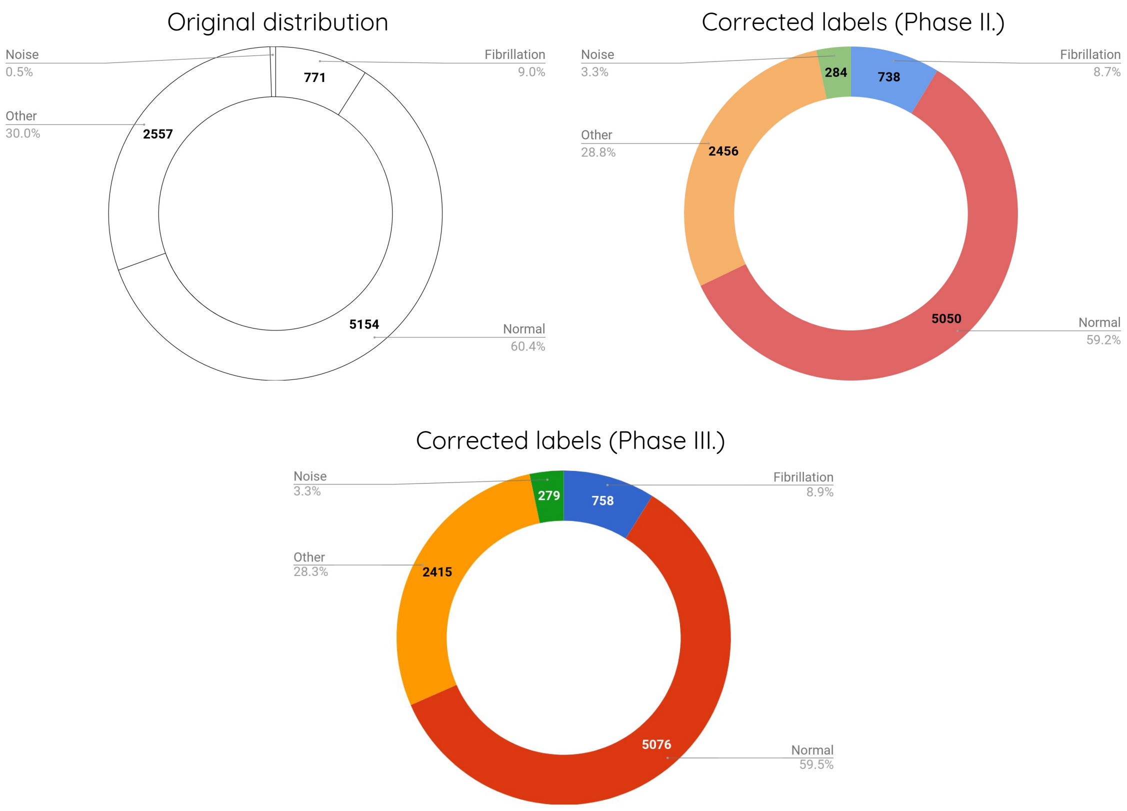


Figure 3.1: During the challenge, the labels have been updated multiple times. The strangest detail was the fact (as it was revealed after the finals), that the organizers used the current top 10 algorithms at each update to determine which samples were the most confusing and should be revised.

Variable length representation

Another challenging detail, is that we have to deal with time dependency. Since we cannot determine how long the samples will be, nor describe a time frame that would fit every relevant pattern, we have to project somehow the sample into a fixed dimensional space. In order to preserve important information by the projection we can separate samples into clusters by previously extracting features which describes best the data. These values are usually time dependent pattern fitting maps. For one-dimensional convolutional networks this would be the generalized idea of [46], visualized on 3.2

Fully Convolutional Networks (FCN)

Based on the fact that hierarchical feature extractors such as the LeNet [42], and so its modern version the VGG [70] are easily trained, reliable, and can be easily restructured for transfer learning purposes. We decided to re-implement the Fully Convolutional Network [46] paradigm in one dimension, as motivated by the following works [51, 40]. The actual convolutional layers of our FCN models are following the pattern:

$$\begin{aligned} c &= \mathbf{K} * x \\ b &= BatchNorm(c) \\ h &= ReLU(b) \\ y &= AvgPool_w(h) \end{aligned}$$

Where BatchNorm is described in [29], and ReLU is the rectified linear unit (nonlinear) activation function, and w indicates the window size of the pooling operation (the stride was set to the same value as well). Hyper-parameters of this convolutional layer are hidden in the dimensions of $\mathbf{K} \rightarrow$ `[IN, OUT, k]`, where `IN` is fixed, `OUT` represents how many feature maps will be created by the convolution operation, and `k` is how large the kernel will be, i.e. how large the receptive field will this layer have on x. These layers were assembled in the depicted manner on Figure 3.3 (top row).

Residual Networks (ResNet)

We also began experimenting with Deep Residual Networks [21], suggested by [79]. The basic idea is simple: we concatenate several of the above mentioned FCNs after each other, while stabilizing the gradient flow by adding the convolutional blocks' input to their output. However the interpretations of why deep residual networks actually work so well are not clear. An excellent article on this topic van be found at[75]. Oversimplifying, we could say each FCN blocks (in Eq. 1) in the model are no longer responsible for detecting global features, but expected as shallow networks to work together as different ensembles,

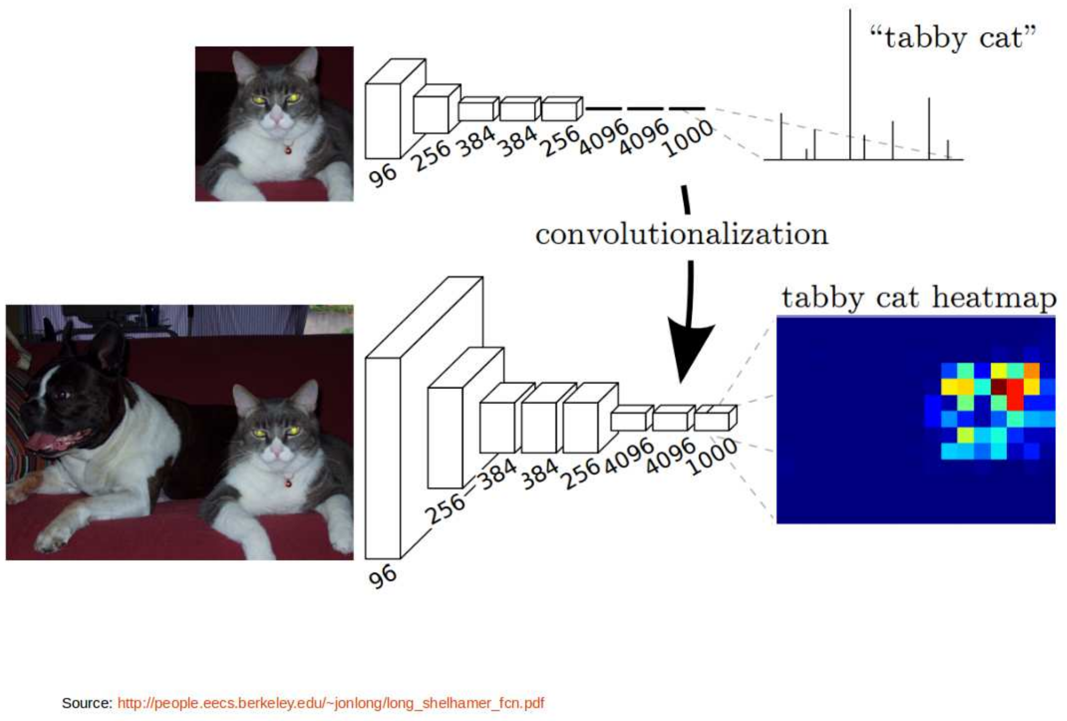


Figure 3.2: Semantic segmentation (bottom) uses the feature extractors from VGG (top), by simply reshaping the fully connected layers into convolutions with local receptive fields. We expect the same behavior when training convolutional networks on one dimensional time series. Image credits [46]

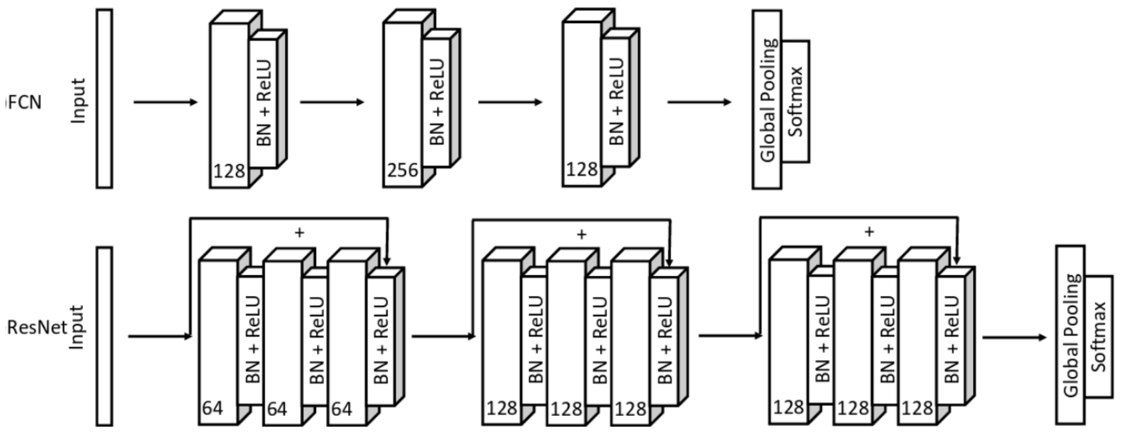


Figure 3.3: A strong baseline provided by Wang et al. for general signal processing tasks. On their benchmark they report that the fully convolutional and residual networks can outperform traditional approaches. Image credits [79]

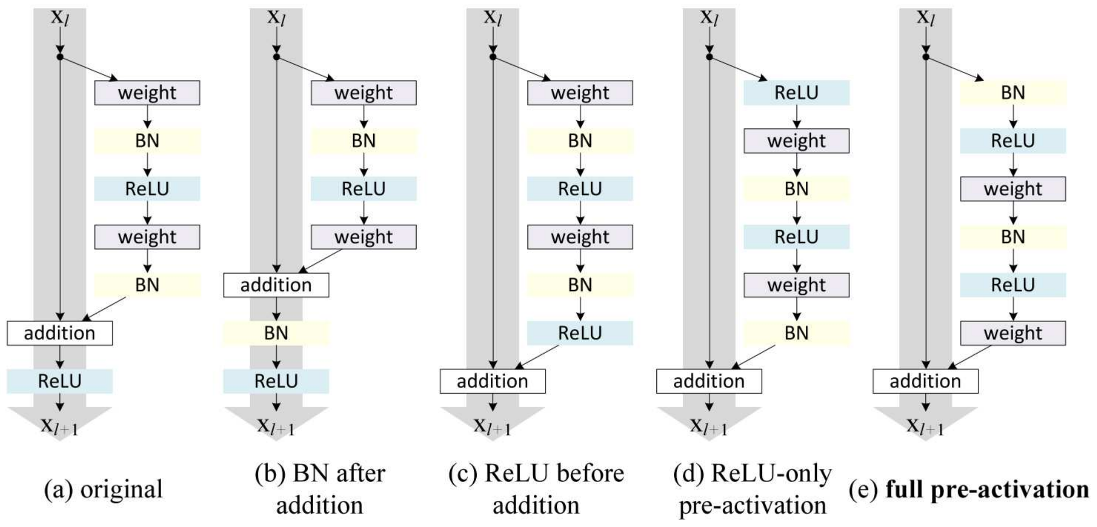


Figure 3.4: Studies of He et *al.* show which sequence of layers in the underlying implementation of residual blocks are the most efficient. For our experiment we applied both the original and the proposed pre-activation ordering, and we can agree with the effectiveness of the latter as well.

which allows us to increase the overall capacity of the model. The ordering between the layers of each residual block and the implementation of the skip connections, especially at pooling steps can be tought of as an additional hyper-parameter of the network - luckily studies of He et *al.* [22] of such delicate differences revealed the best implementation of ResNets. For visual description and alternatives see Figure 3.4

Fixed length representation

Once every filters have been evaluated in the end we can use different approaches: to take the weighted norm or mean of the filtered signal to represent each pattern's presence in the input signal or we can use techniques that have been proven to be successful in natural language processing tasks: recurrent neural networks (RNNs).

Long Short Term Memory networks (LSTM)

Our first attempt was to use a Long Short Term Memory network (LSTM) [24, 49], shortly we switched to its modified version Gated Recurrent Unit network (GRU) [10], but we experienced that the network was not capable to overfit the train set, and achieve a satisfactory error rate. It turned out that other teams applied LSTMs as well, however they stacked it on top of pretrained convolutional feature extractors.

Due to the large number of task specific hyper-parameters in LSTMs and GRUs the search space would exponentially increase, so we decided to continue experimenting with simply reducing the feature map to a single scalar value by mean averaging the activation through time of the last layer of the model, usually referred as the *logit* layer.

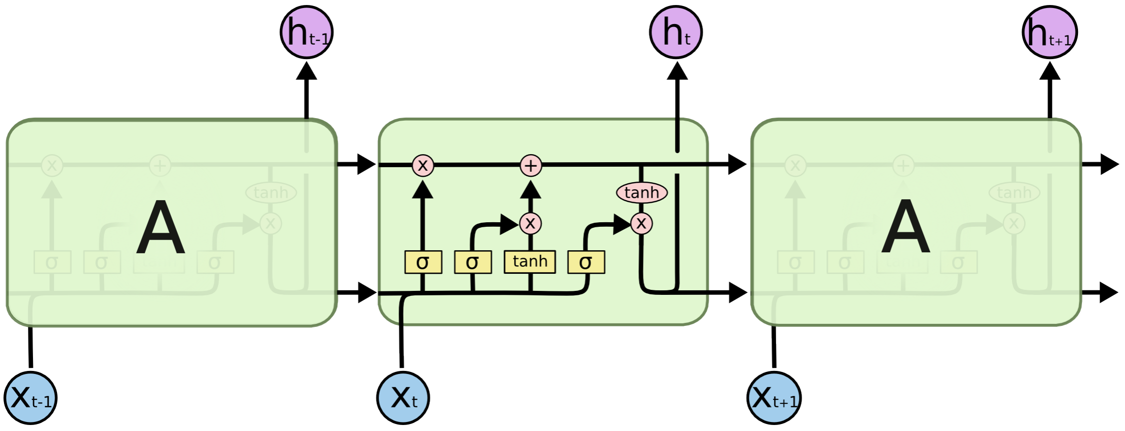
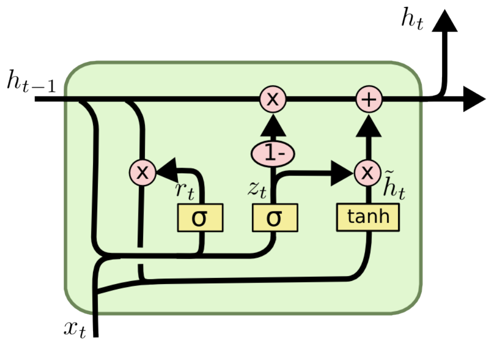


Figure 3.5: Long Short Term Memory network maintains two state, one hidden, and one output state, and via forget and input gates it offers a solution for training recurrent neural networks with back-propagation through time without exploding/vanishing gradients. Image credit: Chris Olah's blog [54]



$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$

$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$

$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$

$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

Figure 3.6: Gated Recurrent Unit networks combine forget and input gates into a single update gate. By discarding the output cell state, the result is a simpler model, that is easier to initialize, and train. Image credit: Chris Olah's blog [54]

Classification

After extracting the most relevant time features from the sample, and reducing it to a fix length feature vector, the model has to separate the feature space with respect to the classes. A basic approach is to apply logistic regression where each feature dimension is taken into account and weighted to determine each class (by dissecting the feature space with hyper-planes). As a result a four dimensional vector $[0, 1]^4$ is produced, where each dimension represents the confidence of the network whether the sample is belonging to the corresponding class or not. The output is then normalized with the softmax function to give a confidence distribution over the classes:

$$p_i = \frac{\exp(y_i)}{|\exp(y)|_1} \tag{3.1}$$

As later will be described we experimented with applying Multi-Layer Perceptrons (MLP) [19], on top of these learned representations, however the results show that the model is more likely to collapse during training-time due to falling in a local minimum.

Advanced methods

Dilated convolution

Evaluating the initial 1D re-implementation of the famous ConvNet and ResNet architectures which are originally intended to work on fix-sized (224 x 224) images of, we realized that the final logit neurons have too small receptive field. Technically that means, with 300 Hz sampling frequency, that the network was required to output a prediction based simply on less than a second - so in order to explicitly adjust the filter's field-of-view whithout collapsing the time-relevant features (i.e. using pooling operations) we augmented the models with a method called Atrous or dilational Convolution [82], such technique was applied for signal generating with WaveNet. For visual supplementary on 2D see Figure 3.7, for the WaveNet reference see Figure 3.8 We tried different patterns, and involved the sequence of dilations as a hyperparameter in our search.

SELU

We have seen the latest updates from the Klambauer et al. who presented Self normalizing neural networks [35] during the time of the contest. They report excellent results on shallow neural networks that are using an alternative non-linearity called SELU derived from Self-normalizing + Exponential Linear Unit layer. The implementation of the activation function is written in Eq. 3.2 where λ and α parameters are the hyper-parameters of the layer. A rare phonemenon in field of recent updates on activation layers, that the *optimal* value of these parameters is provided in [35]. Replacing Rectified Linear Units (ReLU) had an incredible improvement on our architectures as well.

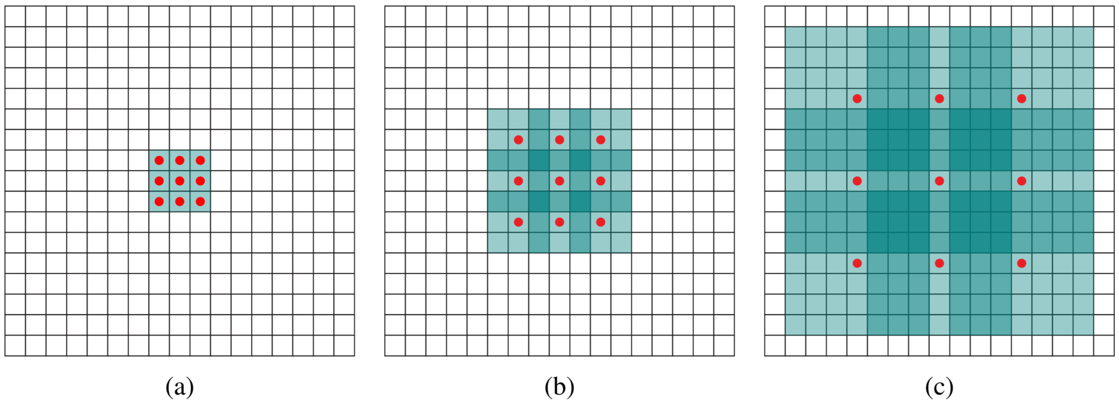


Figure 3.7: As stated in [82]: " Systematic dilation supports exponential expansion of the receptive field without loss of resolution or coverage. (a) F_1 is produced from F_0 by a 1-dilated convolution; each element in F_1 has a receptive field of 3×3. (b) F_2 is produced from F_1 by a 2-dilated convolution; each element in F2 has a receptive field of 7×7. (c) F_3 is produced from F_2 by a 4-dilated convolution; each element in F_3 has a receptive field of 15×15. The number of parameters associated with each layer is identical. The receptive field grows exponentially while the number of parameters grows linearly. "

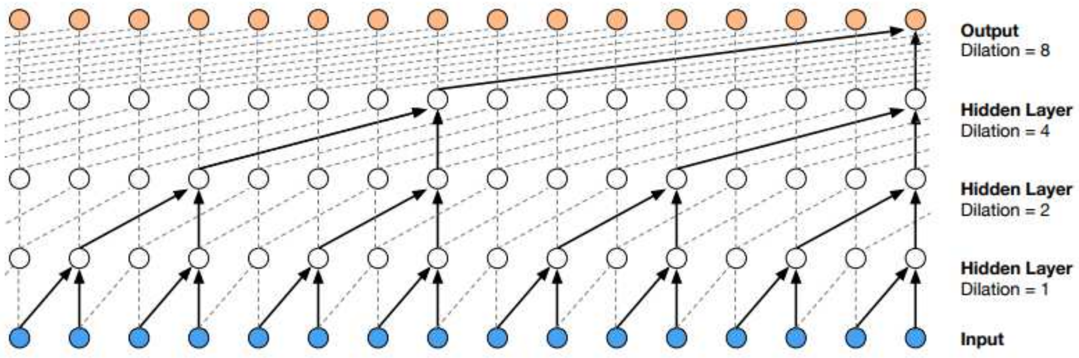


Figure 3.8: Dilated convolutions used for generating realistic audio samples [55] - "layers have various dilation factors that allow its receptive field to grow exponentially with depth and cover thousands of timesteps". Dilated convolutions were a perfect match for our goal to improve the performance while keeping the capacity of our network on the minimum.

$$\mathrm{selu}(x) = \lambda \begin{cases} x & \text{if } x > 0 \\ \alpha e^x - \alpha & \text{if } x \leq 0 \end{cases} \tag{3.2}$$

$$\lambda^* = 1.0507 \sim$$

$$\alpha^* = 1.6732 \sim$$

Representation learning on different domains

We explored the possibilities using both log-spectrogram *and* time domain features.

Time relevant features in frequency domain. We applied a sliding window on the input samples, where each window had been point-wise multiplied by the Hamming weight window and transformed to frequency domain using Real Fast Fourier Transform`RFFT`. In order to increase the sample's variance we took the absolute value of the result and applied natural logarithm element-wise. This resulted in a frequency representation of the original signal, that preserved the time invariant patterns. This sequence of non-linear transformations are usually referred as taking the *logspectrogram* of the sample, and is frequently used in signal processing. We are aware of the fact that during the process, actually by taking the magnitude of the output of the RFFT we discard the information of the phase, however this loss will be later covered by precise choice of parameters of the sliding window.

Matching the dimensions. Solution was to resample the input for the network working on the spectral domain in order that after the spectrogram transfer it had the same length as the input, but with much more channels (freq bands if you like) on frequency domain a VGG19 alternative was the best solution: we implemented the VGG19 without the 2x4096 FC on the end (those layers made every single network worse as my experiments shows...) with special skip connections - after the first 2 layers every layer's output was concatenated to the 2nd layers output (so not in a residual manner). The dilation parameters are the same here on time domain the network mentioned before was applied their outputs were concatenated and a single logit layer was operating on them, with an additional Batch Norm layer before. The logits were aggregated by mean reduction - the same as we did on the CNN.

Augmentation and transfer learning

Despite the initial difficulties with RNN, we began to experiment as a spinoff project to apply RNN as a time series predictor, training it to predict the following N values of the measurements based on the previous parts of the sample. We believe that this model can be reused to initialize training classifier recurrent networks, instead of random

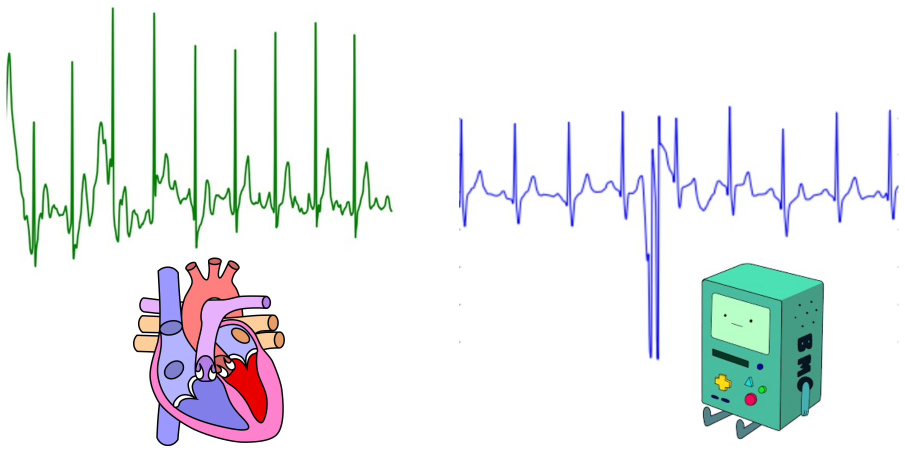


Figure 3.9: Artificial sample produced by an LSTM trained to predict and continue original ECG samples. Left: an AF Challenge 2017 entry classified as *normal*. Right: ECG sequence produced by a network trained on exclusively *normal* samples. In this case the network functions as some sort of language model of ECG. Among many samples, professionals could not tell whether samples produced by this method were artificial or taken from the original training set. Truth on being told, after further analysis they could detect artifacts, still some said that a very ill patient could produce the same samples. Image credits: Wikimedia, Adventure Time.

initialization. On the other hand, by feeding back the network its own predictions we could produce data 3.9. The network in this case would function as some sort of "language model" [50] of ECG signals - and if trained properly, i.e. on a single class, it would output labeled, yet totally artificial samples, which could be used as augmented training data as well.

4. Evaluation

In this chapter we describe the details of the parametrized function we optimized to map ECG samples to cardiac failure classes. We have tried many of the available alternatives, and ideally would list all of our previous experiments for others to avoid the pitfalls we have come across - however, due to limits we list those elements that were used in the most successful architectures with our initial motivation that led us to these solutions.

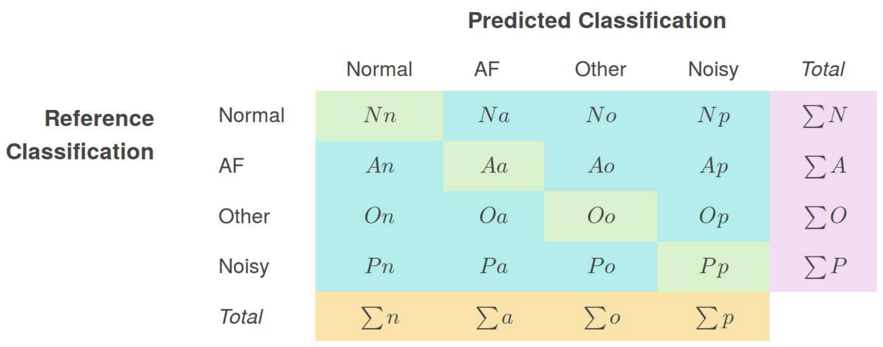
Test environment

Evaluation

The best practice on evaluating the classifier model's performance is to simply compare the number of matching labels evaluated on a completely disjunct set of samples from the training set. However, in our case it was not so trivial to score different methods, since by simply answering *normal* to every sample would yield 50+% success rate, therefore it would be less representative.

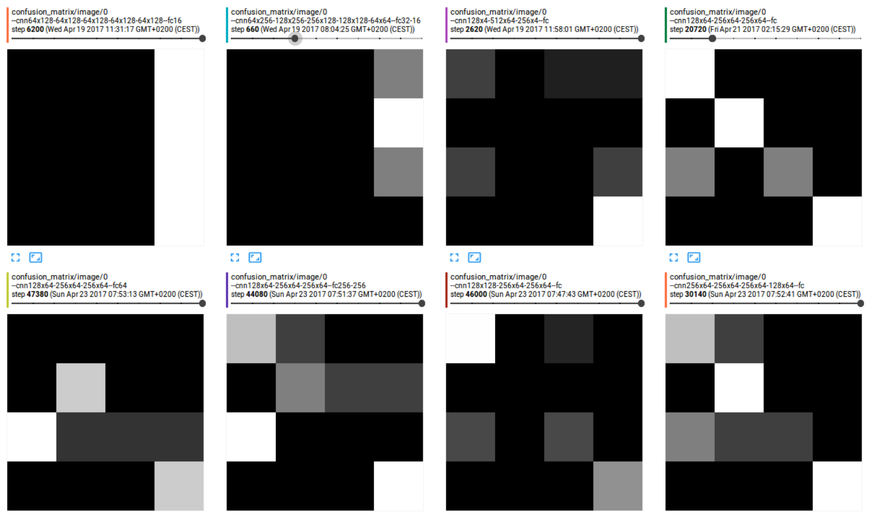
Confusion matrix. Instead we are using an extended version of precision-recall evaluation, namely we apply a *confusion matrix*, where every row represents a histogram of the correct label, and every column represents the predictions of our model. The diagonal elements show how many labels match, but it preserves the distribution between classes. Using this method, we can easily track down when a training routine collapses, and the network is only using a few of the available classes. When the diagonal elements over number the off-diagonals, then the network has been successfully trained. For examples of the confusion matrix see Figure 4.1. By reducing the matrix to a single scalar defined on the website of AF Challenge, we get an accuracy value, which tells us the exact score we would obtain by submitting an official entry on behalf of our research group. In the final phase of the competition the official evaluation metric has changed.

Class balance issue. After the competition has ended, we realized, that our implementation of the evaluation metric had the following bug: in every iteration we processed the whole evaluation set, however the input producer function used to assemble the batches applied the random oversampling method as well.



		Predicted Classification				
		Normal	AF	Other	Noisy	*Total*
Reference Classification	Normal	Nn	Na	No	Np	$\sum N$
	AF	An	Aa	Ao	Ap	$\sum A$
	Other	On	Oa	Oo	Op	$\sum O$
	Noisy	Pn	Pa	Po	Pp	$\sum P$
	Total	$\sum n$	$\sum a$	$\sum o$	$\sum p$	

Figure 4.1: Confusion matrix. Diagonal values represent prediction **hit**, off-diagonals represent **miss**. Image credits AF Challenge 2017



```
confusion_matrix/image/0
--cnn64x128-64x128-64x128-64x128-64x128--fc16
step 6200 (Wed Apr 19 2017 11:31:17 GMT+0200 (CEST))

confusion_matrix/image/0
--cnn64x256-128x256-256x128-128x128-64x64--fc32-16
step 660 (Wed Apr 19 2017 08:04:25 GMT+0200 (CEST))

confusion_matrix/image/0
--cnn128x4-512x64-256x4--fc
step 2620 (Wed Apr 19 2017 11:58:01 GMT+0200 (CEST))

confusion_matrix/image/0
--cnn128x64-256x64-256x64--fc
step 20720 (Fri Apr 21 2017 02:15:29 GMT+0200 (CEST))

confusion_matrix/image/0
--cnn128x64-256x64-256x64--fc64
step 47380 (Sun Apr 23 2017 07:53:13 GMT+0200 (CEST))

confusion_matrix/image/0
--cnn128x64-256x64-256x64--fc256-256
step 44080 (Sun Apr 23 2017 07:51:37 GMT+0200 (CEST))

confusion_matrix/image/0
--cnn128x128-256x64-256x64--fc
step 46000 (Sun Apr 23 2017 07:47:43 GMT+0200 (CEST))

confusion_matrix/image/0
--cnn256x64-256x64-256x64-128x64--fc
step 30140 (Sun Apr 23 2017 07:52:41 GMT+0200 (CEST))
```

Figure 4.2: Confusion matrix visualized in TensorBoard for different runs with Fully Convolutional Networks

Train environment

The training set

In order to keep track of how well our model generalized the features, and to avoid over-fitting, the original downloaded samples were separated into two sets, into **train** and **evaluation** set at an 80-20 ratio. We re-sampled randomly selected entries to make an even distribution between classes. We use evaluation set to keep track how our model performs on unseen data.

Data standardization

We experimented with different normalization techniques such as standard normalizing per sample, or with the global mean and average with no significant difference in the results. Our current experiments involve standardizing the samples by heart rate frequency, thus the class relevant invariance can be easier learned by the the network.

Toy problem. A simple example is the following: suppose we have two patients, one who drinks coffee regularly resulting in a high heart rate (even in healthy) and one who runs a marathon every week with low heart rate at rest. If both were having the same cardiac symptom probably the cardiologist would record their natural heart rate to have a baseline, a reference point to use for diagnosis. This is because the time domain patterns are expressively specific to the patient's heart rate. Technically this means, if the atrial fibrillation specific curve would appear in both samples it would probably appear elongated or compressed — and the model had to learn both patterns as if no similarity would have exist between them. This problem could lead to a situation where the network would discard some rare feature, in order to reserve the capacity for the trivial pattern just with different lengths.

Solutions. To counterweight we could increase the capacity of the network, which often leads to significant improvement in performance on the train set, however at the same time as a by-product the performance on the validation set decays, since the model is more likely to over-fit, simply memorizing the train samples, and loses its ability to generalize well. The loss is likely to happen since our training set only contains less than 8K samples. An other solution is standardizing the heart rate frequency of our samples by resizing the whole sequence in time, which would in case, solve our toy example. We made sure that discarding the sample specific frequency has no significant relevance to the corresponding class. For details of class BPM variance see Figure 4.3

Weighted loss. Our first attempts to improve accuracy on rare samples was to set higher loss on underrepresented classes to encourage the network to optimize by learning

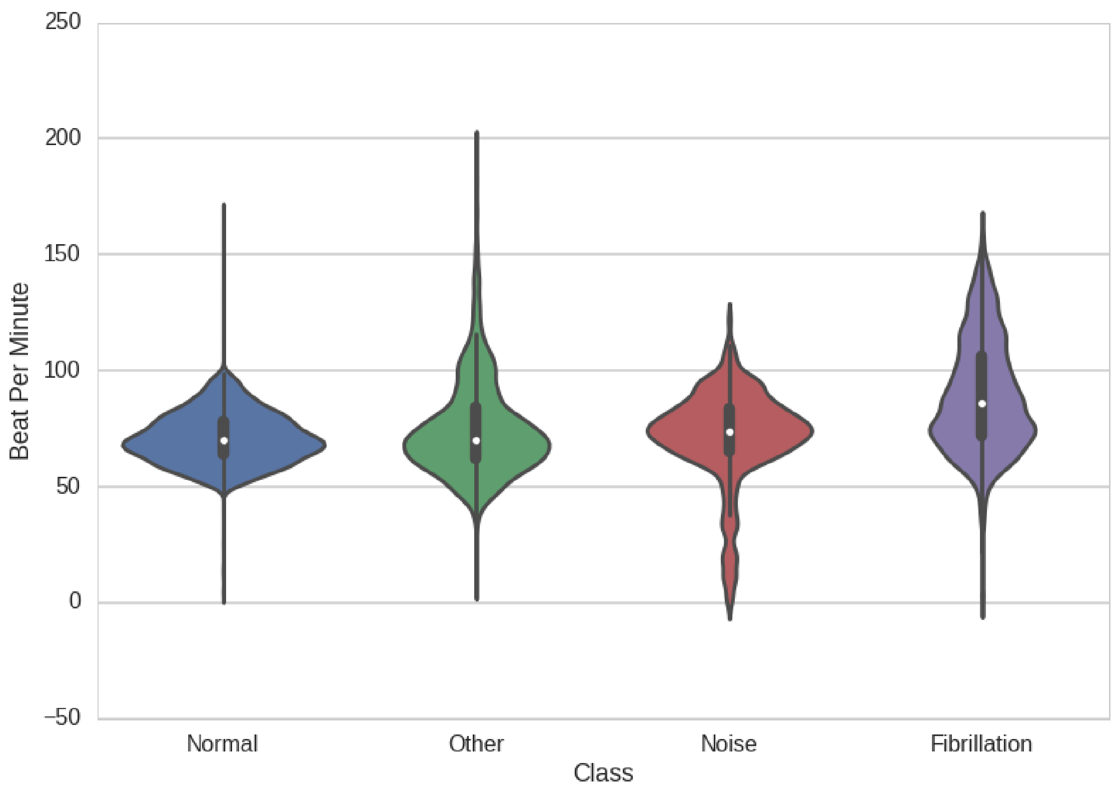


Figure 4.3: The heart rate distribution and variance between classes is similar (except of cases in Noise class where the heart beat detection algorithm fails to find the R waves, thus resulting in extreme values, which we intend to filter before the entry reaches the valid sample classifiers), so discarding the average heart rate by normalizing every sample by the value, presumably would not affect negatively the performance of our network.

to recognize these samples better, but it resulted in over-fitting and soon was replaced by augmented evenly distributed mini-batches.

Mini-batch. Mini-batch training and evaluation is a common method throughout almost every Deep Learning algorithm. It enables the back-end to feed the network with multiple samples at once. Both parallel inference and weight adjusting is implemented in the latest machine learning frameworks for mini-batch processing.

Default parameters

Capacity. As a baseline we initialized all our experiments with the same number of channels as stated in the reference architecture, except that instead the usual $N \times N$ kernels we used a $N^2 \times 1$ kernels for convolution operations.

Optimizer. We have experimented with the basic Stochastic Gradient Descent method comparing against the current state-of-the art optimizer algorithm ADAM [33].

Regularization. DropConnect [77] with $p = 0.5$ settings. We also apply weight decay, by adding the L_2 norm of each θ in the network's parameters to the overall loss function and use soft labels (perturbed one-hot vectors) in order to prevent the networks from favoring a specific class. Our training policy varies over different settings, currently we are using early stopping method: after 10 consecutive steps where validation performance has not improved the trainer shuts down, to enable other train routines use the allocated device — otherwise the models are evaluated after a previously given number of steps. Throughout the training, we decrease the learning rate, in order to prevent the model from oscillation.

Time domain classification

The literature research we made suggested, and the cardiologists helping us confirmed that doctors make diagnosis according to the time domain representation of the recording, so we put strong emphasis on researching that area, thus the majority of our experiments were carried out in time-domain. The main advantage of classification over time domain that by applying visualization methods such as inspecting the activation maps at each layer, or advanced techniques like guided backpropagation [81] and deconvolution [84] helped us to inspect specific patterns recognized by the algorithm. During these experiments, the data preprocessing steps also evolved with the different architectures we tried.

We trained and evaluated the following models in time-domain, with at least 2 cross validations and 420 epochs per each - the listed results are the mean of the corresponding approach.

Initial experiments

LSTM. At first, we carried out multiple experiments on single LSTM-s with different width (100, 150, 256, 512, 1024) and number (1, 2, 3) of stacked layers. For all of the experiments, we used ADAM optimizer without adapting learning rate and dropout. Mean and variance was always shifted to $(0, 1)$ as a preprocess step, and the length of the training samples was a fixed to 100, 1000, or 3000.

The test results were quite disappointing: Only the LSTM with 256 width 2 depth, trained on long (3000 points, 10 sec) samples was able to reach 0.64 train F1 score, but even that model performed dreadfully having only 0.32 F1 score on the hidden test set. However, that network was fast thanks to the TensorFlow implementation: Running the network on a single k80 GPU yielded 210 sample/sec training speed and 450 sample/sec forward speed.

LSTM on CNN. Afterwards, we essayed to combine LSTM with CNN, hoping that LSTM can transform variable length features (e.g. QRS complexes) to fixed length feature vectors that are good inputs for CNNs. Thus, we kept 2 layer of LSTM, and trained them on a CNN from [79] under identical training settings to the previous experimenlts. The performance of the model was still not satisfying, so we inserted batch normalization layers between convolutional layers to the first network that performed over 0.80 on train set.

We submitted that network as our first entry containing working neural networks, and its official F1 score on the hidden test set was 0.72. (We already had had two preceding entries: One was a test entry guessing normal class for every sample that yielded 0.15 as F1 score, and the other one utilized only hand-crafted feature extractor and an SVM over it that scored 0.45 on hidden test set.) Nonetheless, the performance of that architecture still did not meet our expectations, it was much more efficient in classification and just slightly slower (Training speed of the network was 170 sample/sec, and forward speed was 370 sample/sec running the TensorFlow implementation on a single k80 GPU).

Time dependency and variable length. Dealing with variable length samples can be done in multiple ways:

1. With random-cropping we get fixed length samples, therefore length of time-dependent features coming out from CNNs will be deterministic.
2. We can deploy an RNN on the top of the variable length feature vectors and after they've processed them, the hidden state will hold the information retrieved from the sequence in a fixed state vector. Sadly our experiments with training CNNs and RNNs in an end-to-end fashion have failed, due to instability.
3. We can apply mean reduction for different length feature vectors and a linear classifier on the average of each feature for the whole sequence.
4. We can evaluate the predicted class for each time instance and take the average of the prediction. This was the most successful way of dealing with varying length samples: it works more robustly, and over-fit is less likely to occur in the logit-reducing scenario.
5. We can use variable sized windows: Time dependent (different length) feature vectors could be processed at once with a fully connected classifier on the top of it, however that means, that the weights of each specific time frame should be learned separately. In that sense, shifting our training sample by one arbitrary time-frame all the time-dependent learned features would be discarded. To avoid that we came up with the idea of using fix number of different length windows for reducing the length-variant samples to a fixed length representation (by hand, instead of using an LSTM for the same purpose). So when we would feed a 9000 and a 18000 long input, our feature vector would be the length of 9 and 18 (supposing our feature extractor is scanning 3.3 seconds with a stride of 3.3 seconds - 1000 sampling points) and after applying the variable sized window, for example $N = 3$ both sample would yield a feature vector of length 3 (for each feature extracted by the CNN). In other

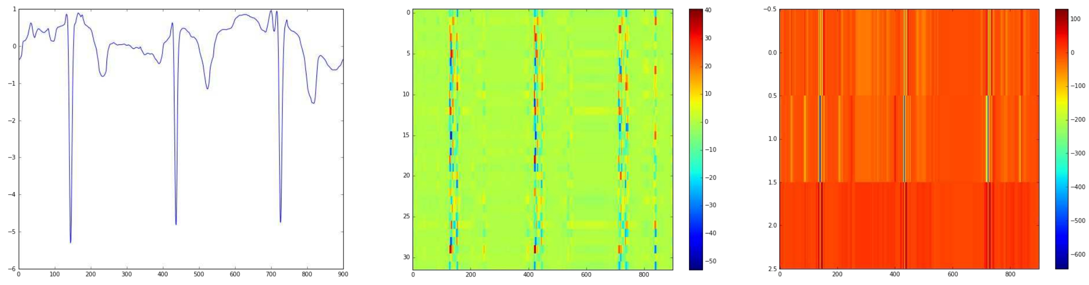


Figure 4.4: A crop from a sinus-rhythm sample can be seen on the left. The activation map of the feature extractor in the middle (extracting 128 features in total) clearly increases magnitude when the R wave is scanned - so do the logit layer that performs a single linear transformation on the previous layer for each time frame - resulting in class suggestions based on the amplitude of each R wave. This immediately meant that we need to increase heavily the receptive field of each layer.

words, with $N = 3$ every sample would end in a matrix that has F columns (each representing whether the sample contained the corresponding pattern or not), and 3 rows (whether the feature was present in the beginning, in the middle and in the end of the sample).

First we tried the variable sized windowing method, but these experiments were rather instructive than ground braking, mostly suggesting that one should not use variable sized windows. We tried also the others, and finally, we decided on mean reduction, but we still had two options where to perform that reduction: A) We could average the feature vector before the logit layer, or B) evaluating the logits at multiple time instants and mean reducing the final choice of the network. Later it turned out that option B) performs better, so we used in our later models.

During the debugging of the flawed system we also learned, that feature extractors in our succeeding experiments were theoretically useless. The three layered CNN, with each layer having a kernel size 17×1 turned out to be scanning only a $3 \times (17 - 1) = 48$ window - at most. This theoretical barrier only became clear when we visualized the output of the last layer, see on Fig 4.4

We implemented the model in TensorFlow, and because there is no recurrent part in the network, it became much faster. (Running on a single k80 GPU the training speed is 1200 sample/sec, and the forward speed is 2110 sample/sec.)

ImagNet models. After some further literature research, we found a couple cutting edge architectures that might fit our needs: VGG16, VGG19, ResNet, Pre-Activation ResNet, WideResNet. We tried them applying the same modification to all: Instead of using $N \times N$ kernels, we used $N^2 \times 1$ kernels to keep the overall capacity of the network. Initial results were good, and soon we began tuning the preprocess steps to push the limits of these networks. Running the PyTorch implementation (we switched to PyTorch because we found that prototyping is much faster for us in PyTorch compared to TensorFlow) on a single k80 GPU, results of these networks were the following:

- VGG16
  - F1 score: 0.81 (train set), 0.74 (evaluation set)
  - Training speed: 570 sample/sec
  - Forward speed: 990 sample/sec
- VGG19
  - F1 score: 0.83 (train set), 0.75 (evaluation set)
  - Training speed: 430 sample/sec
  - Forward speed: 870 sample/sec
- ResNet-50
  - F1 score: 0.81 (train set), 0.75 (evaluation set)
  - Training speed: 340 sample/sec
  - Forward speed: 540 sample/sec
- WideResNet-28-10
  - F1 score: 0.82 (train set), 0.79 (evaluation set)
  - Training speed: 210 sample/sec
  - Forward speed: 300 sample/sec

Hand crafted features. We also tested some of the most important wavelet features, and R-R variability indices computed externally. Sadly the features completely confused the logit neurons, resulting always in worse performance when random Gaussian noise was used instead of features (since the weights just simply decayed to zero for the noise). Tested on three VGG16, VGG19 and ResNet-50, it reduced their overall test F1 to: 0.62 0.60 0.59 (respectively)

Increasing receptive field. Just after the previous experiments, we noticed the presence of the CNN receptive field problem. In the previous trials we tried larger pools for max pooling: [size:stride] 10:1, 10:2, 20:2, 100:50. From these experiments we learned that while increasing these parameters can improve the train performance, on the downside it also increases the gap between the train and the test error since the network is more capable to overfit on massively downsampled data. Consequently, we started using Dilated Convolutions to overcome the receptive field problem, and we applied a custom sequence of dilation parameters for each test. Results show that using dilated convolutions in the low level feature extractors is more effective than in the latent layers. For

Residual Network we applied dilation only for the first convolutional layer, while the second layer always had a dilation parameter of 1 (i.e. no dilation applied).

Late experiments

We trained 134 networks in time domain, and ideally we would share the details of each experiment, because one could find general rules that apply elsewhere, but due to space constraints we only list our most instructive experiments from the last weeks of the challenge.

Preprocessing. For these experiments, we added two preprocessing steps: One is randomly multiplying the samples by -1. because sometimes R peaks were the most negative values in the sample - so we had to train a sign invariant model. The other was the 2.2 sigma thresholding because without that the dilated convolutional networks would have been still driven by the magnitude of the sample. In order to reduce this effect we just chopped of the top of the r-peaks to see whether it was working or not. Sigma 2.2 had been performing better than other sigma values (1, 2, 3, 5, inf) validated with 5-fold cross-validation on two networks: WRN-28-10 and VGG16.

Training. To make the training set keep the same class imbalance, we randomly selected 90 percent of each category. We tried both 256 and 512 batch size, and we used 300 epoch for every network. Besides, we tried mwo optimizers, and we experienced that ADAM performed significantly better than SGD when initial learning set is set to 1e-4 and decreased linearly to 0 over the training session.

Architectural changes. Also we experienced significant improvement by removing the dense layers (2 fully connected layers that are usually the last two layers before the logit layers in ImageNet models) which mainly combined the activation maps of convolutional layers because their capacity appeared oversized (usual width from 1024 to 4096). In a task like ImageNet classification with 1000 categories, they probably make a good choice, but on our problem set they blocked the gradient flow.

Finally we learned that by using SELU layers, described at Eq. 3.2, even if the training error was higher, the distance between the test and train accuracy was smaller than in cases of ReLUs. Also the phenomenon of "dead neurons" (neurons that are constantly outputting zero not effected by the input) was avoided by using SELU

Hierarchical classification. During the contest, the scoring system has changed: the final score was the mean of the F1 scores of normal, AF, other classes. Noise class was not directly evaluated (however it still affected the precision of the detection of other classes). To maximize our score, we trained a binary classification network as a first step simply to detect noisy samples. When the sample was not detected by the noise discriminator network, than we fed it in to a three-way classifier. We tried further hierarchical alternatives, but they did not yield significant improvement. Our experiments showed that even a shallow network could perform over 90% on noise detection, and by this approach we hoped that the noise discriminator capacity trained separately could reserve more generalization ability of the 3-way classifier.

Custom architectures.

Encode-Net Inspired by the network used for fast neural style-transfer [18], we designed a network with the following layers:

- convolution (32) + batch normalization + SELU + max pooling
- convolution (64) + batch normalization + SELU + max pooling
- convolution (128) + batch normalization + SELU + max pooling
- convolution (256) + batch normalization + SELU + max pooling
- dilated, preactivation Residual Block (256)
- dilated, preactivation Residual Block (256)
- dilated, preactivation Residual Block (256)

where convolution layers had dilation factor of 2 and no bias, and kernel size of 11, 7, 7, 7 respectively, in the residual blocks the second convolutional layer was dilated with the factor of 2, and each residual block had two convolutional layers with kernel size of 11 and 9 respectively. Residual blocks' inner architecture depicted on figure 3.4, and the schematic structure of our proposed feature extractor can be seen on figure 4.5.

We implemented that model with PyTorch and got F1 scores of 0.98 on train set and 0.92 on evaluation set. Running on a single k80 GPU, the training speed was 630 sample/sec, and forward speed was 1120 sample/sec.

Skip Fully Connected Network (SkipFCN) Motivated by densely connected CNNs [27] we refactored our VGG16 network in a manner where the output of the first pooling layer was concatenated to every succeeding pooling layer. (Many other architectures were tested as well with no significant improvement, however this network later turned out to be super-effective on spectral domain). For details see figure 4.6.

That that architecture was implemented also in Pytorch with slightly lower results: F1 scores was 0.93 on training set and 0.88 on evaluation set. Running on a single k80 GPU the training speed was 560 sample/sec and forward speed was 970 sample/sec.

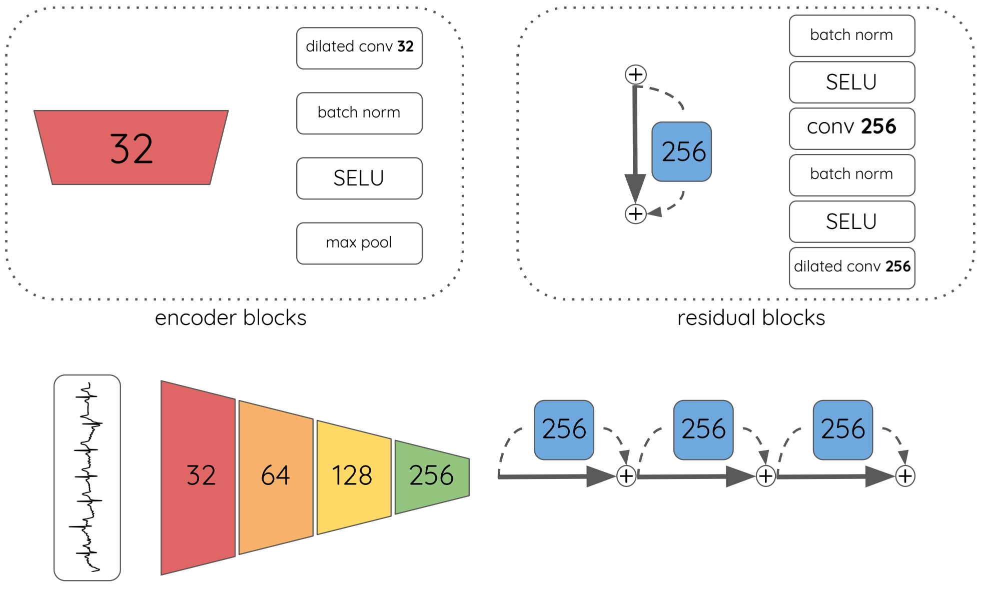


Figure 4.5: Encode-Net Temporal dimension is gradually reduced while increasing number of feature maps provides broader/wider connection with the residual network, thus keeping computationally inexpensive. Hierarchical feature maps are trained more robustly, while the residual layers can operate more efficiently on latent representations of these high level features.

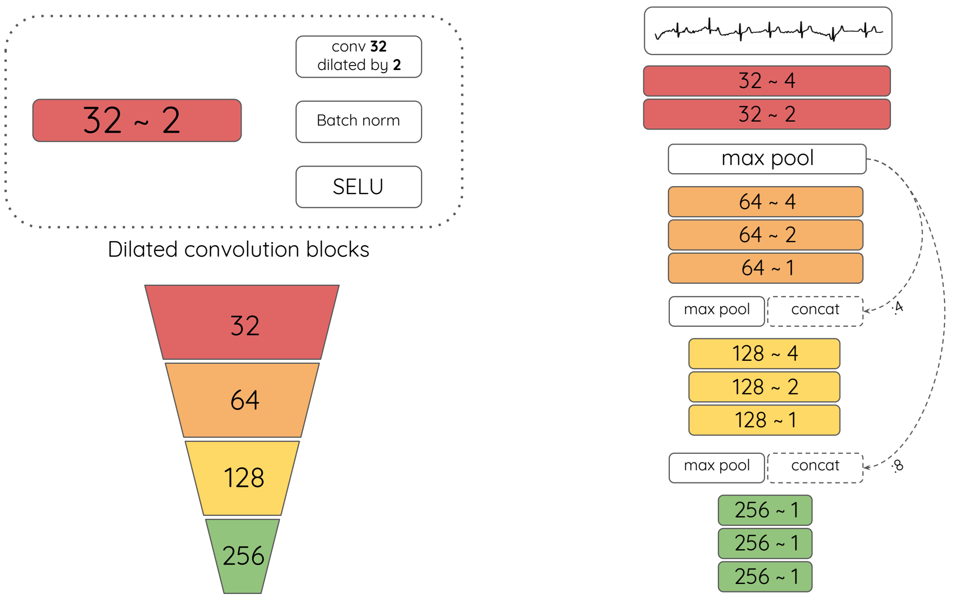


Figure 4.6: SkipFCN: A classic fully convolutional VGG16 network without the three dense layer of 4096 neurons, using 9×1 convolution kernel instead of 3×3 and halved each layer's capacity. On time domain the Encode-Net outperformed this architecture, however on spectral domain the skip connections improved the overall generalization by breaking the symmetry.

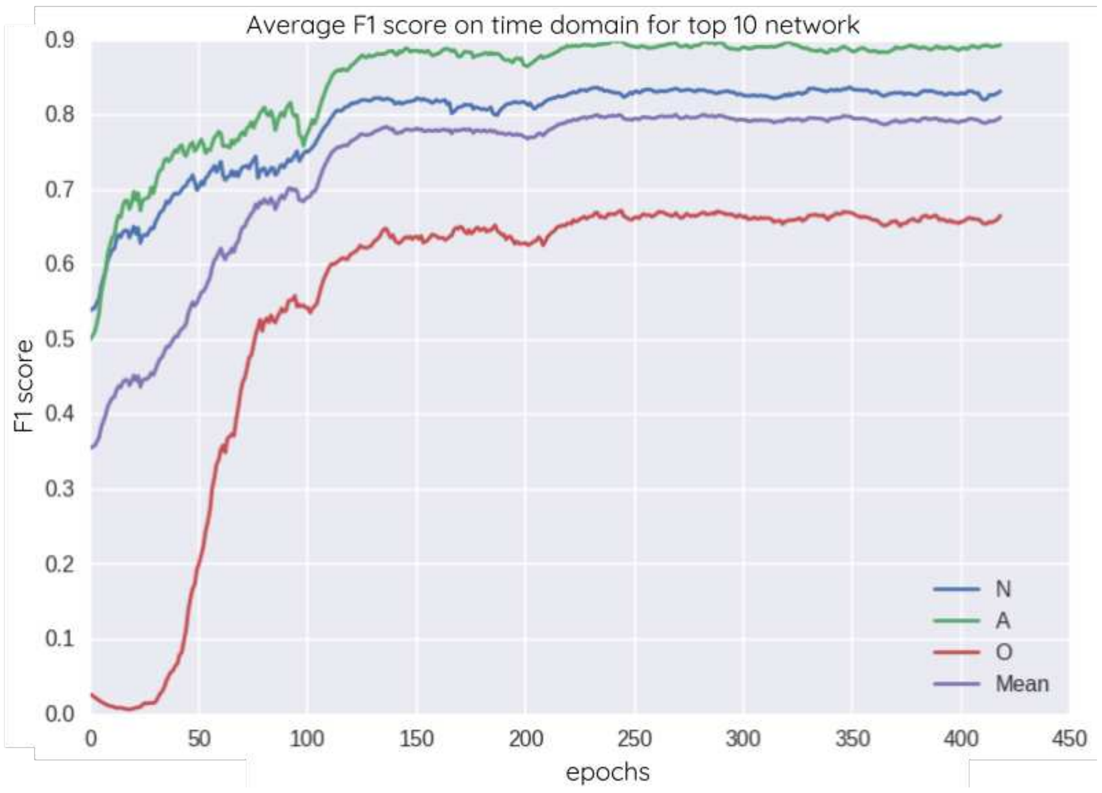


Figure 4.7: Taking the average of the top 10 performing network revealed that our detectors that operate in time domain (almost independently from the underlying architecture) tend to perform excellently on Atrial Fibrillation category, while the worst score was the F1 of the *Other* class.

Conclusions

Finally we concluded that networks operating in time-domain are the most efficient in detecting *AF*, while the *other* class is always a drawback for the overall accuracy. See figure 4.7

Frequency domain classification

Initial experiments

Initial experiments were built on complete sequences, by applying FFT to the whole sequence. It turned out that the class invariant features were too delicate, and were diminished by taking the Fourier Transform of the complete signals.

Late experiments

Skipping to the last phase of the challenge, we experimented with training networks that were successful on time domain, with the difference that now we used the logspectrogram of the samples.

Log-spectrogram. By taking the FFT of smaller windows we can analyze the spectral components of the sample and preserve temporal features at the same time. Our experiments with the fine-tuning of different parameters of taking the spectrogram of samples concluded to using **stride** parameter of 1 for the sliding window, Hanning weight before applying FFT, and $(2N + 1)$ window size for retrieving $N = 32$ number of channels from the transformation. It is a general step afterwards to apply elementwise logarithm of the absolute value on the spectrogram to increase its variance. *Important note:* This step discards phase and therefore the transformation will be irrevertible, still it improved the performance of the network in almost all cases. For a detailed visualization see figure 4.8. We tested the last two architectures, we created in the time-domain experiments, and results on a single k80 GPU are the following:

- freq-EncodeNet
  - F1 score: 0.95 (train set), 0.88 (evaluation set)
  - Training speed: 450 sample/sec
  - Forward speed: 540 sample/sec
- freq-skipFCN
  - F1 score: 0.94 (train set), 0.93 (evaluation set)
  - Training speed: 410 sample/sec
  - Forward speed: 530 sample/sec

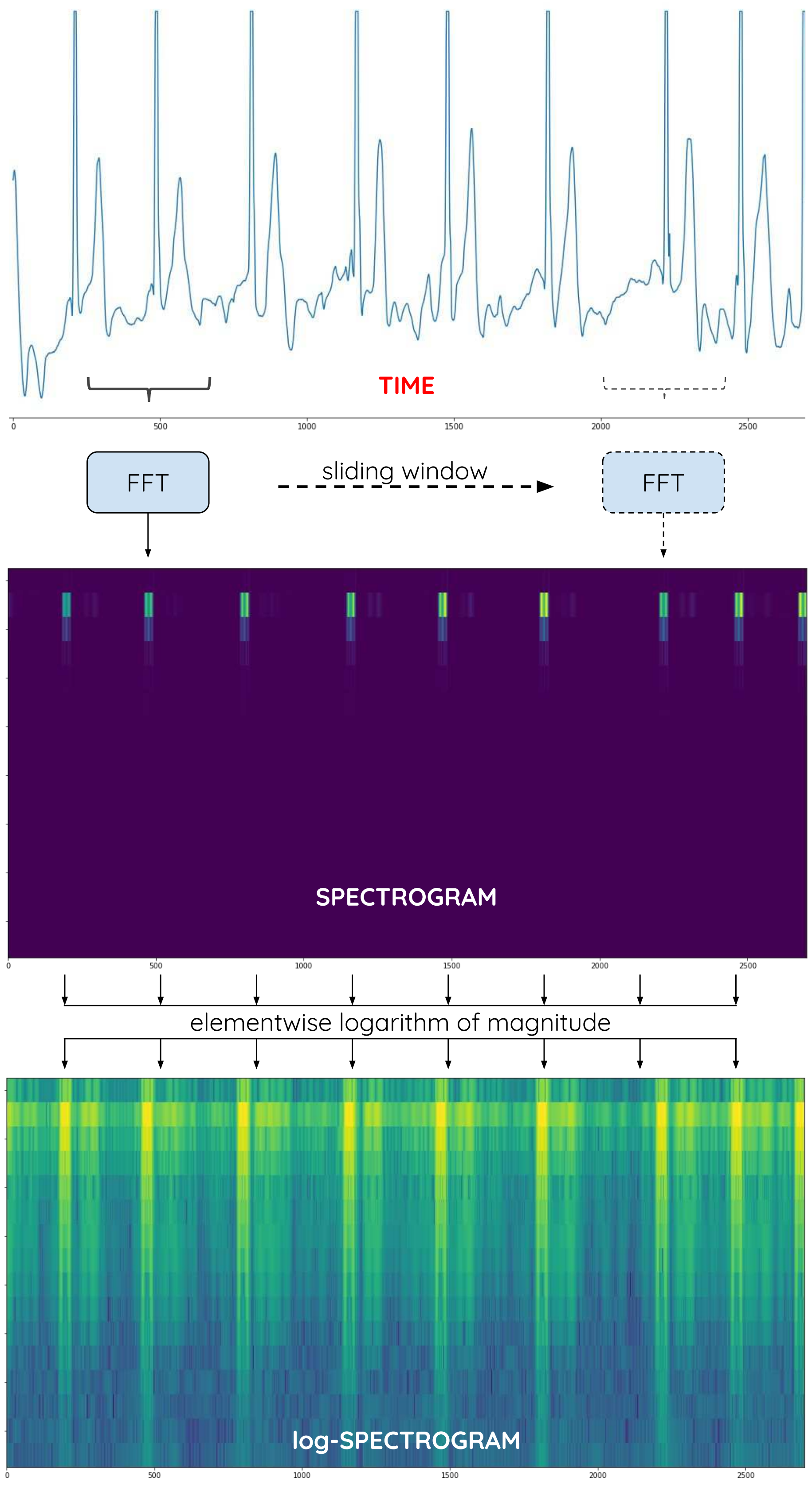


Figure 4.8: Preprocess step that provides spectral domain representation while preserving time-domain patterns. Using a sliding window of size (2N+1) with stride of 1 yields an almost identical length representation as the input sample but in N number of channels - making the signal highly redundant, yet incredibly informative for the classifier algorithm.

Multi-domain classification

In the last two days of the challenge we have been able to implement a forked preprocess technique that was built on the most successful attempts' experience, taking the following steps:

- Sample $N/4$ number of random samples from each class, with $N = (batchsize) = 16$.
- Random crop a 2400 section from the samples.
- Fork the pipeline to **time** preprocess
  - Random multiplying the samples by -1 to increase the robustness of the model towards inverted records
  - Threshold the peaks between $\sigma = 2.2$ for each sample
- Fork the pipeline to **spectral** preprocess
  - Take the spectrogram of the random crop with window size of 65, Hanning window weight, overlap of 64
  - Resample with Nearest Neighbour method the 32 channel output to match the length of the original time crop: 2400
  - Take the absolute value piecewise
  - Take the logarithm piecewise

Late experiments

We cut straight to the chase by only selecting the best performing network on each domain to run our trainings on.

- Time: EncodeNet F1 - 0.92
- Freq: SkipFCN F1 - 0.93

The settings of our last experiments contained the forked preprocess steps and both model initialized with Xavier [37] technique.

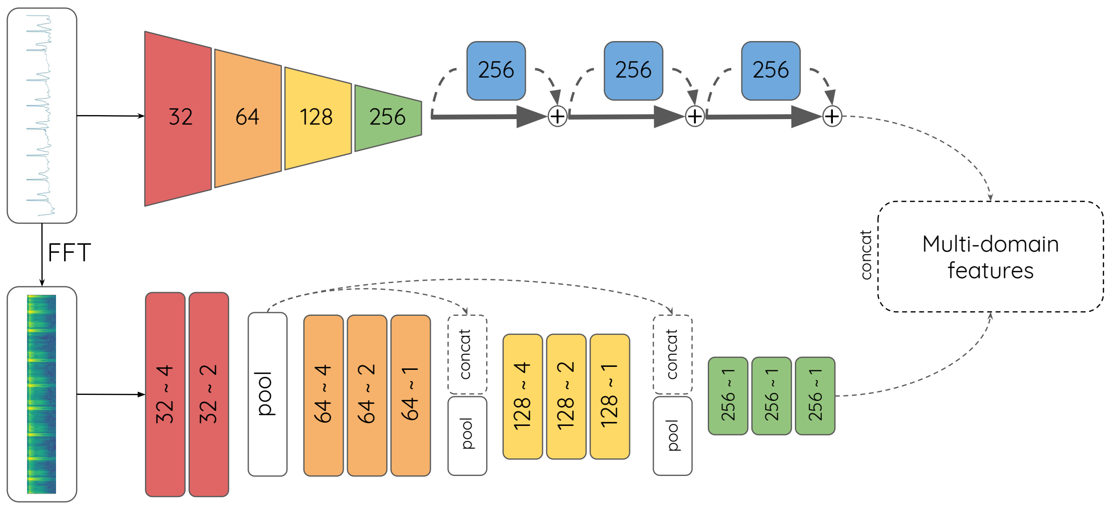
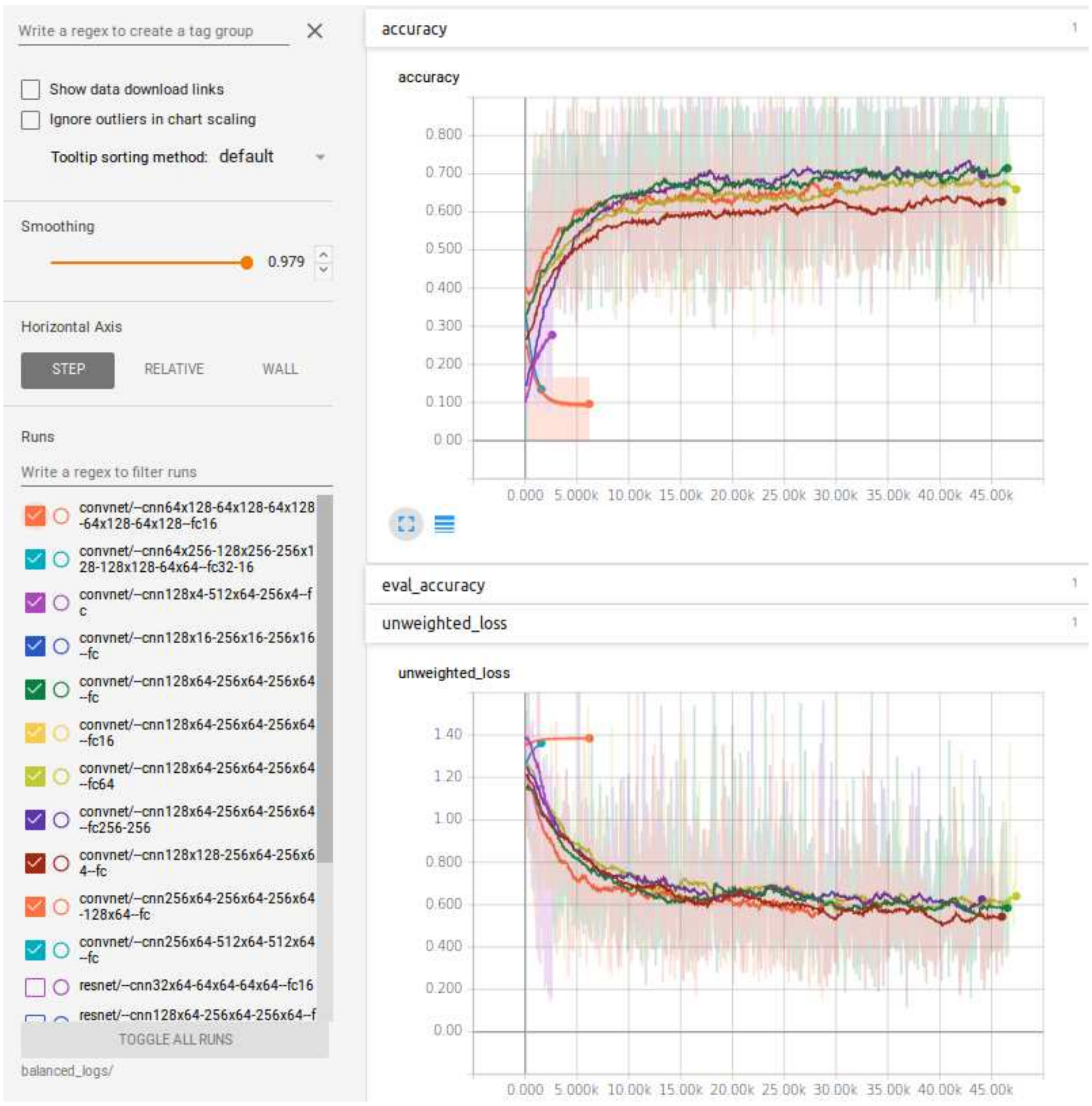


Figure 4.9: Processing a single-lead ECG signal on both time and spectral domain with our proposed architecture. To be able to concatenate the spectral activation maps to the temporal features we re-sampled the signal with nearest-neighbour method.



Write a regex to create a tag group

☐ Show data download links
☐ Ignore outliers in chart scaling

Tooltip sorting method: default

Smoothing 0.979

Horizontal Axis [✓ STEP] RELATIVE WALL

Runs

Write a regex to filter runs

☑ convnet/--cnn64x128-64x128-64x128-64x128-64x128--fc16
☑ convnet/--cnn64x256-128x256-256x128-128x128-64x64--fc32-16
☑ convnet/--cnn128x4-512x64-256x4--fc
☑ convnet/--cnn128x16-256x16-256x16--fc
☑ convnet/--cnn128x64-256x64-256x64--fc
☑ convnet/--cnn128x64-256x64-256x64--fc16
☑ convnet/--cnn128x64-256x64-256x64--fc64
☑ convnet/--cnn128x64-256x64-256x64--fc256-256
☑ convnet/--cnn128x128-256x64-256x64--fc
☑ convnet/--cnn256x64-256x64-256x64-128x64--fc
☑ convnet/--cnn256x64-512x64-512x64--fc
☐ resnet/--cnn32x64-64x64-64x64--fc16
☐ resnet/--cnn128x64-256x64-256x64--f

TOGGLE ALL RUNS

balanced_logs/

Figure 4.10: Overall performance of the FCN approaches plotted in TensorBoard. Top: Accuracy derived from the confusion operator. Bottom: Unweighted loss during training

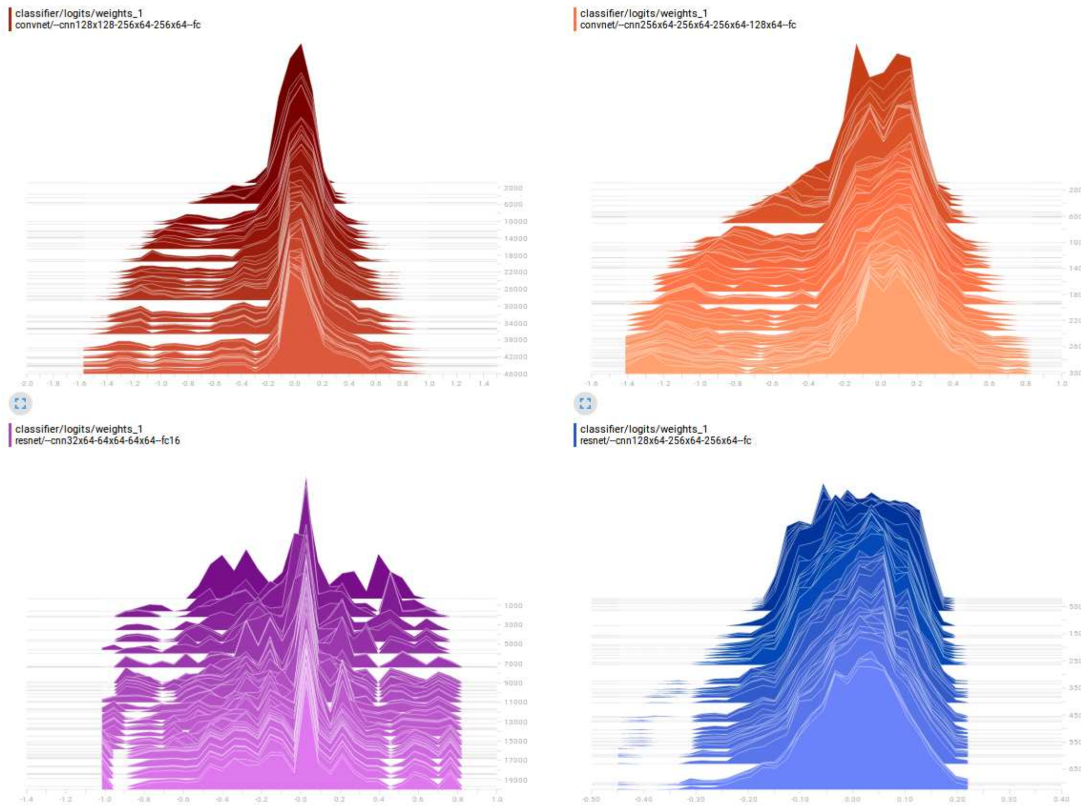


Figure 4.11: Weight distribution of the classifier (last) layer through training. Note that networks with small variance, and local edgy peaks are more likely to fall into a local minimum of the parameter space — where choosing the same class no matter what is the input is too stable state for the network to learn any further features.

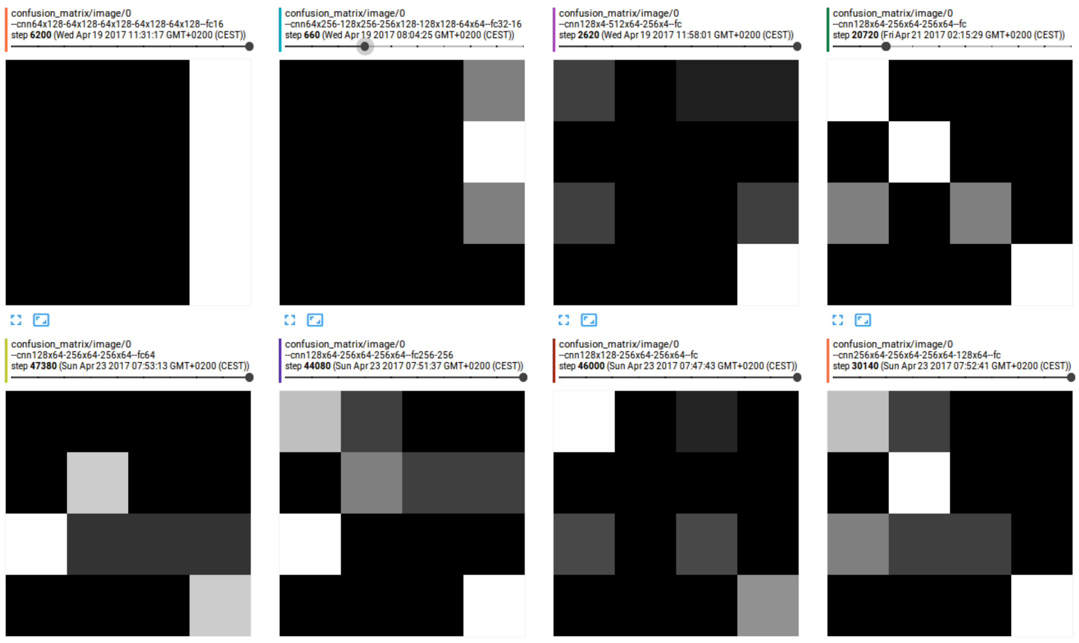
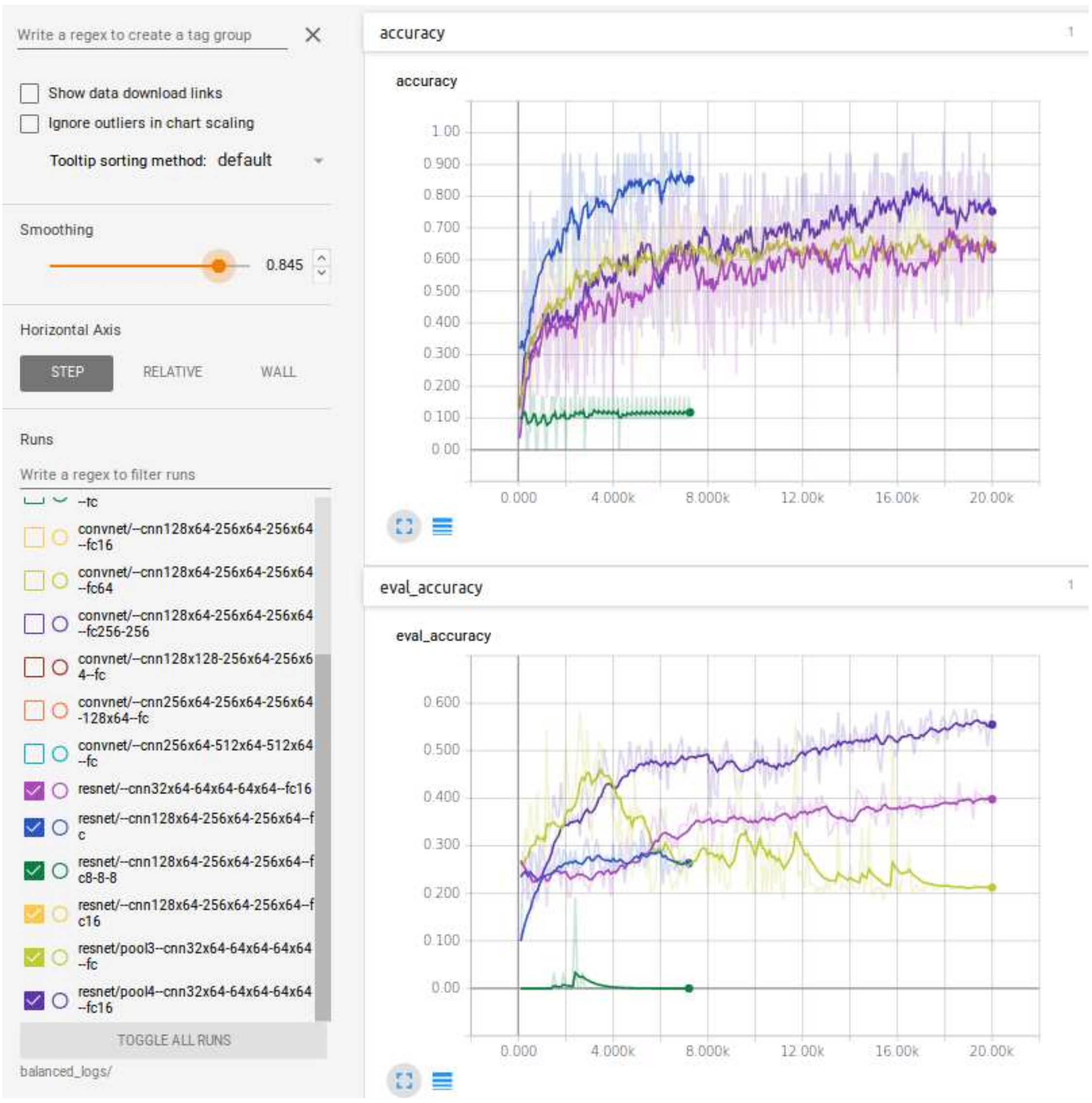

confusion_matrix/image/0
--cnn64x128-64x128-64x128-64x128-64x128--fc16
step **6200** (Wed Apr 19 2017 11:31:17 GMT+0200 (CEST))
confusion_matrix/image/0
--cnn64x256-128x256-256x128-128x128-64x64--fc32-16
step **660** (Wed Apr 19 2017 08:04:25 GMT+0200 (CEST))
confusion_matrix/image/0
--cnn128x4-512x64-256x4--fc
step **2620** (Wed Apr 19 2017 11:58:01 GMT+0200 (CEST))
confusion_matrix/image/0
--cnn128x64-256x64-256x64--fc
step **20720** (Fri Apr 21 2017 02:15:29 GMT+0200 (CEST))
confusion_matrix/image/0
--cnn128x64-256x64-256x64--fc64
step **47380** (Sun Apr 23 2017 07:53:13 GMT+0200 (CEST))
confusion_matrix/image/0
--cnn128x64-256x64-256x64--fc256-256
step **44080** (Sun Apr 23 2017 07:51:37 GMT+0200 (CEST))
confusion_matrix/image/0
--cnn128x128-256x64-256x64--fc
step **46000** (Sun Apr 23 2017 07:47:43 GMT+0200 (CEST))
confusion_matrix/image/0
--cnn256x64-256x64-256x64-128x64--fc
step **30140** (Sun Apr 23 2017 07:52:41 GMT+0200 (CEST))

Figure 4.12: Models are monitored through the training process: during weight updates a small subset of unseen entries is inferred, and compared to the ground truth labels. In these *confusion matrices* the row sum represents a histogram of the labels, while the column sum would represent the histogram of the network's choice. Their intersection results in the talkative heat maps. Notice that those models whose confusion matrix is mainly diagonal is performing well, and those networks which have been collapsed gives a matrix where only a single column can be seen (i.e. first row, the two rightmost matrix).



Write a regex to create a tag group

☐ Show data download links
☐ Ignore outliers in chart scaling

Tooltip sorting method: default

Smoothing 0.845

Horizontal Axis: STEP RELATIVE WALL

Runs

Write a regex to filter runs

--fc
☐ convnet/--cnn128x64-256x64-256x64--fc16
☐ convnet/--cnn128x64-256x64-256x64--fc64
☐ convnet/--cnn128x64-256x64-256x64--fc256-256
☐ convnet/--cnn128x128-256x64-256x64--fc
☐ convnet/--cnn256x64-256x64-256x64-128x64--fc
☐ convnet/--cnn256x64-512x64-512x64--fc
✓ resnet/--cnn32x64-64x64-64x64--fc16
✓ resnet/--cnn128x64-256x64-256x64--fc
✓ resnet/--cnn128x64-256x64-256x64--fc8-8-8
✓ resnet/--cnn128x64-256x64-256x64--fc16
✓ resnet/pool3--cnn32x64-64x64-64x64--fc
✓ resnet/pool4--cnn32x64-64x64-64x64--fc16

TOGGLE ALL RUNS

balanced_logs/

Figure 4.13: Overall performance of the ResNet approaches plotted in TensorBoard. It is worth to mention that the network represented by the green curve is using the same hyper-parameters as the network with the highest train performance (blue), still because of the MLP block it fails to learn a generalized representation. Another important feature is that the network represented by the blue line performs way better on the training set, still fails on the evaluation set. This is the classical example of an over-fitted model. Top: Accuracy on the training set. Bottom: Accuracy on the evaluation set.

5. Medical Relevance and Visualization

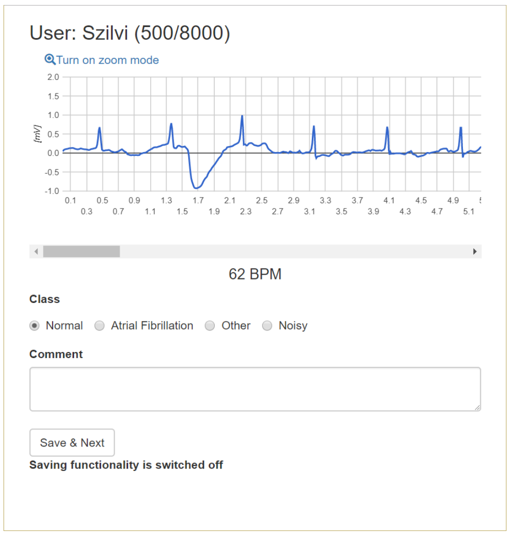
Besides evaluating how well the classifier model performs on the training and test set, we wanted to examine how relevant our results are medically. To achieve that, we contacted two cardiologists to help us: Dr. Szilvia Herczeg and Dr. István Osztheimer, both working at the Heart and Vascular Center of Semmelweis University. Our goal was first to compare the decisions of the cardiologists and our model, then we wanted to explore which are the most important features the model is looking for.

Quality of data set

Method

Website First thing we did to allow the cardiologist to contribute to our project was designing a website that displays the recordings and provides a graphical user interface to annotate the displayed recordings and leave comment under the interesting recordings to make the annotation more informative. That way we obtained an alternative annotation that helps us to validate the data set; i.e., what are the obvious cases, and which recordings are insufficient to make a firm diagnosis. The website handles multiple users, and it even has a minimalist registration form to register new users.

Design and features. When making a front-end for the website, we paid attention to make the design simple and clear, yet to give all the information that can help classification, so only the graph of a single recording and the belonging input fields (radio buttons for the class labels and an optional comment field) is visible to the user when he or she annotates data. The chart is scaled to the standard format originated from traditional paper-based ECG recordings (the ECG graph paper is moved at speed of 25mm/sec, and the paper is divided into 05x05mm grid-like boxes that represents 200ms at time scale and 0.5mV at y axis). The main feature of the charts that they are zoomable, so the user can observe even small details of the recordings. Additionally, the average BPM (beat per minute) is indicated under the graph to aid decision-making. The website picks the recording randomly, but the distribution of the chosen recordings over the classes is uniform, meaning that the user get a recording from each class with equal probability.


User: Szilvi (500/8000)

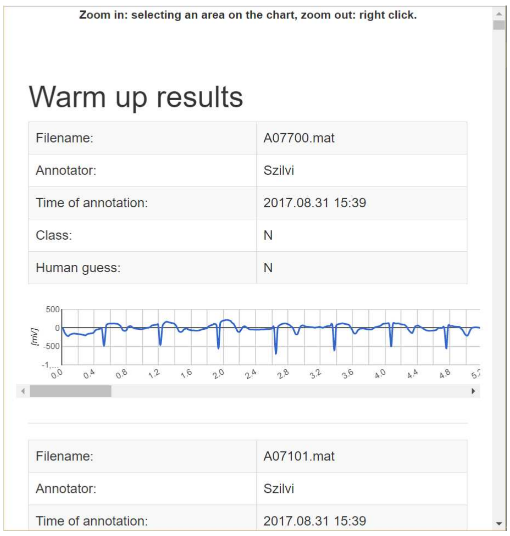
Class

[✓ Normal] Atrial Fibrillation Other Noisy

Comment

Saving functionality is switched off

Figure 5.1: Screenshot of "test" page of the website


Zoom in: selecting an area on the chart, zoom out: right click.

Warm up results

Filename:	A07700.mat
Annotator:	Szilvi
Time of annotation:	2017.08.31 15:39
Class:	N
Human guess:	N

Filename:	A07101.mat
Annotator:	Szilvi
Time of annotation:	2017.08.31 15:39

Figure 5.2: Screenshot of "browsing" page of the website

Website pages. We created two very similar pages for the website: One is for "warm up", meaning that this one is aiming to help the user to get familiar with the interface and the data set, and another one is for "test" – it is used to measure the performance of the user over the classification problem. The only difference between them that the former one tells the user what class does the recording belongs to according to the cardiologists of the challenge, and also it provides another comment field (e.g. for the case if user wants to leave a comment upon possible misclassification). Figure 5.1 shows a screenshot of the layout of the "test" page. Furthermore, we created a web page to browse the annotated recordings. It shows the file name, name of annotatior, time of annotation, labels according to the cardiologist of challenge and our annotator, and the comments. Figure 5.2 depicts that "browsing" page.

Results

Dr. Szilvia Herczeg kindly offered that she annotates a subset of the data set, and accordingly she annotated 500 recordings so far. We compared her annotation against the annotation of the cardiologists of the Challenge, and we found that she agreed the classification only in approx. 65% of recordings. That underlines the fact that classification of a large part of the data set is not evident even for experts. However, when we compared her annotation against the output of our model, it turned out that considering only those cases when she agreed with the cardiologists of the challenge (i.e. the evident cases) the model gave the same prediction for classes as the cardiologists. Even more surprisingly, apparently Szilvia's annotation has slightly more matches to the model's prediction than the annotation of the cardiologists of challenge (though it can be an artifact due to the low number of annotations). For more information see table 5.1.

Label	All	Correct	Match	Correct & Match
Normal	124	111	107	104
AF	97	70	74	70
Other	140	45	63	43
Noisy	139	103	102	102
Total	500	329	346	319

Table 5.1: Result of Szilvia's annotation. Label: class name, All: number of annotated recordings by Szilvia grouped by reference annotation, Correct: number of matches between Szilvia's annotations and the reference annotation, Match: number of matches between Szilvia's annotation and the model's prediction, Correct & Match: all agree. Reference annotation: annotation of cardiologists of Challange

Confidence of the classifier

Methods

Output of model. We wanted to examine not only the data set, but our classifier model as well, so we also created a website to show the recordings that were easiest or the hardest to classify for our model. To achieve that we took advantage of the fact that the output layer (that is also called logit layer) of the network produces numbers in the range of $[0, 1]$ for each class, and each number represents the confidence of the network whether the sample belongs to the corresponding class or not. So we fed all recordings to the classifier and picked the top 10 recordings which had the largest output number for the normal class, and we did the same for the AF class. We assumed that those recordings can also be interesting that were difficult for the classifier, so we selected the 10 recordings that belongs to the least confident decisions for both normal and AF class. We decided to inspect only these two classes because other and noisy classes are only technically necessary for defining the problem, but they have no medical relevance because noisy recordings must be repeated in medical practice and the other class consists of numerous different arhythmias that are treated totally different ways.

Visualization website and evaluation. In order to make the selected recordings visible to the cardiologists, we added a page to the website similar to the previous ones; however, it displays only the graphs of the selected 40 recordings. The graphs are scaled to the standard format and the grid lines over the graphs are also standard, and the graphs are zoomable. We sent a link to that page to the cardiologists, and asked them to try to find some common features between those that are classified into the same class. We also asked them whether the recordings, that the model classified confidently, were evident for them, too, and also the opposite, i.e., whether they found the least confidently classified recordings obscure as well.

Technologies and URL. The website is mobile friendly, and utilizes the following technologies: HTML5, PHP, JQuery, Bootstrap, Google's chart Javascript library, and

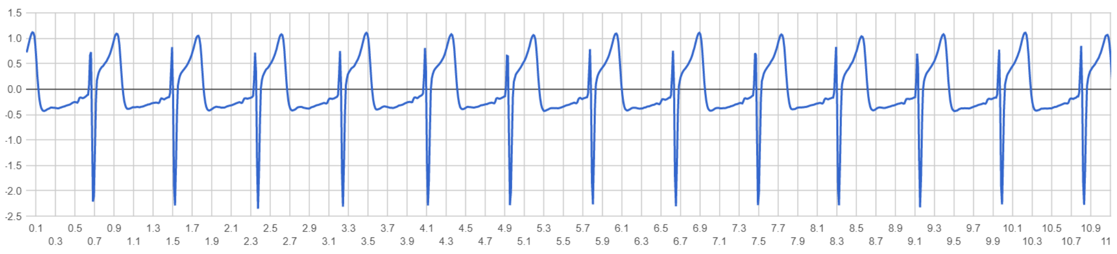


Figure 5.3: Section of one of the recordings that corresponds to the most confident predictions. Unit for x axis is sec, and for y axis is mV.

SQLite. It is hosted by Pázmány Péter Catholic University. The website can be reached at the following URL: `http://users.itk.ppke.hu/~hakta/challenge/`

Results

We asked both Dr. István Osztheimer and Dr. Szilvia Herczeg to look at the selected 40 recordings, tell us whether are the recordings of most confident machine predictions are easy for them or not, and try to find tendencies. We answered them independently from each other, but their remarks were very similar in most aspects.

Noise. First, they both agreed that the in case of most confident predictions the recording contained no or low, distinguishable noise, especially in case of recordings with normal rhythm. Figure 5.3 shows an example for these samples that were easily classified due to the very low noise. Similarily, they both mentioned that the main difficulty in classifying the recordings that were classified by the model with least confidence is the low amplitude noise and irregular baseline changes that makes P wave detection very difficult. In case of hardly detectable P waves, they both would rather look at the RR intervals whether they are regular or not. For more details on cardiac signals see Appendix A.

RR intervals and BPM. In addition, István noted that half of the selected recordings classified as normal with lowest confidence are rather AF recordings, but the unusually regular RR intervals can be very misleading. Moreover, Szilvia noticed that all of the confidently classified normal recordings have arhythmia absoluta, meaning that RR intervals are always changing, and most of these recordings have high BPM value, while recordings of low confidence prediction have much lower BPM on average. She mentioned that the arhythmia absoluta and BPM are two of the most common features cardiologists are looking for in real life clinical practice.

Conclusion. The final conclusion from both of them was that it is true that the recordings that were hard to classify for the model are very difficult for them, and the high confidence predictions are evident for them as well, and the model might detect some of the important medically relevant features.

Most relevant segments of recordings

Interpretation of layers of the model. For further investigations, we had to look inside the model. As it is mentioned earlier in the "Evaluation" chapter, the model overcomes the problem of variable length input by using mean reduction, meaning that it scans through the input with a short window, and the output of these scanned sections are averaged at the very end of the model. For the sake of simplicity, we can image that every neuron gathers information from the recent few milliseconds and fires according to that information in every moment of the recording. Finally the very last layer of the model takes the mean of these time-dependent outputs. Including that layer, we have inspect 3 layers at the top of our model:

- Feature layer: That layer is looking for 256 different features over time (i.e. there is 256 neurons in that layer). It produces a large positive number in a given moment, when looking at the section of the recording from the recent past it recognizes the pattern it is looking for. If it is something very different, it can produce even negative output.

- Sublogit layer: There is 3 neurons in that layer, that take the weighted sum of the output of feature neurons for each neurons. Each neuron weights the features differently, giving high weight for the features that describes the corresponding class, and low weights for those, that does not.

- Logit layer: It produces the output of the model by averaging the output of the sublogit layer over time.

Coloring the graph background.

We were also curious what are the most important segments of each recording that contributes most to the decision of the model, so we took the output from sublogit layer because it tells us which is the most probable class for each short sections of the recording, and colored the background according to that. The advantage of that method is that it even displays how confident the guess was. The classifier for noisy class is a separate model, so the graphs shows confidence values only for normal, AF, and other class. The white color means totally uncertain, green means normal, red means AF, blue means other. The colors are not mixed to avoid abundance of information, so always only most confident guess' color is used. An example for these graphs can be seen on the figure 5.4. We showed that graphs to István, and he concluded that the model might be mainly sensitive to different kind of irregularities, and also the quality of irregularities (irregular patterns can change according to a rule or in a totally irregular manner).

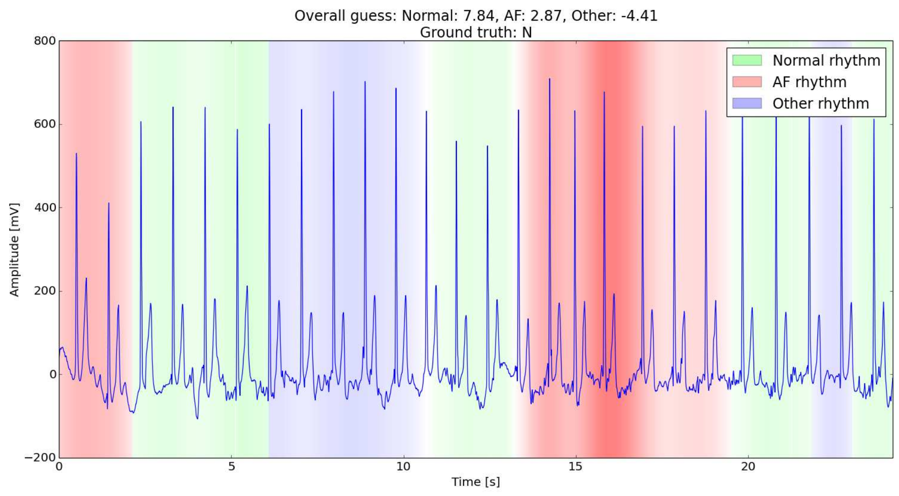


Figure 5.4: Example for graph with colored background. Depth of colors indicates how confident was the classifier for the given section.

Classes against each other.

During preparing the output of the sublogit layer for conversion to RGB colors, we noticed that there might be a correlation between the outputs of sublogit layer for each class over time, that is, for instance when the model produces high value for AF class, then the values for normal and other class will be low with higher chance. Figure 5.5 shows a clean example of that phenomena. Following that guess, we computed sublogit output for each sample, and calculated correlation for all class pairs over time (that way we got 3 values for each recording). Then we took mean and standard deviation of these correlation coefficients over the samples. The results was the following:

- Normal vs. AF: -0.5189 (mean), 0.3299 (std deviation)
- Normal vs. other: -0.0998 (mean), 0.5892 (std deviation)
- AF vs. other: -0.6850 (mean), 0.3442 (std deviation)

It confirmed our suspicion that there is a strong negative correlation between the other and AF classes, and a less strong negative correlation between normal and AF class, and the standard deviation is relatively low for these two comparisons, while there is no such a strong correlation between the sublogit output of normal and other classes and the deviation from the mean value is also much higher in that case.

Weights under sublogit layer.

To discover the phenomenon described above, we had to dig a bit deeper to our model: we inspected the weights coming from the feature layer to the sublogit layer. From the 256 features of our model, and we picked the top 50 feature neurons for each class that contributes most to the given class, that is, the feature neurons that have the strongest connection (highest weight) with the sublogit layer neuron of the corresponding class. We

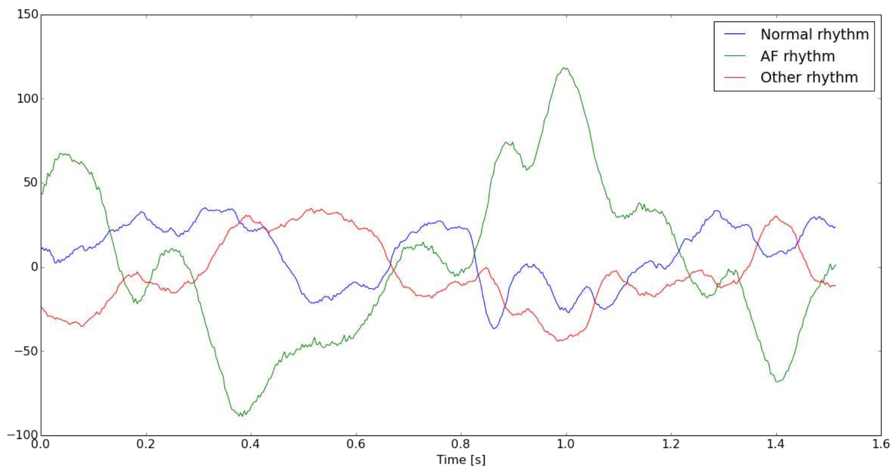


Figure 5.5: Sublogit layer output of model for the same recording as at the figure 5.4. That graph is a good example for "competing classes" because as AF value goes higher, the other two classes value go down, and vice versa.

did the same for the most negative weights, then we matched these subsets of features against each other. The table 5.2 shows the result of that matching, that is, how many shared features are between classes. While it might be confusing that 25 + 1 shared features where normal and other class are "against each other" and yet there is no correlation between the logits of the two class, it turns out that there are some shared features that suppresses of promotes normal and other together. That also can be a possible explanation why the classifiers have always worse performance on the class other than either normal or AF.

Further experiments.

We wanted to visualize somehow the features the model is looking for, so we tried multiple approaches.

Best fragment for strongest features. First, we selected the top 10+10 of the most positive and most negative features, cut the samples into at most 1.7 sec long pieces, and computed the logit values for all samples of training set. Then, we looked up the top 10 fragments with highest logit value for each classes for each features, and plotted them hoping that we can notice some common patterns, but no tendency was visible.

Optimizing input. Second, we attempted to optimize only the input layer to produce a "perfect" sample that makes the model decide as confident as it is possible. We tried both producing "perfect" normal sample, and producing a "perfect" sample for one of the features. In the former case, we set the goal as optimizing the input layer until the logit layer produces $[1, 0, 0]$ as output, and in the latter case, the goal of the optimizer was to approach the optimal output of feature layer that is a vector that consists of only zeros except one places that selects the desired feature. These trials was also unsuccessful

Positive for...	Negative for...	Number of features
AF	other	37
other	normal	25
other	AF	11
normal	AF	11
AF	normal	1
normal	other	1
normal, other	-	1
normal, other	AF	4
AF	normal, other	1
normal	-	33
AF	-	22
other	-	9
-	normal	23
-	AF	24
-	other	11

Table 5.2: Distribution of shared features over classes.

because apparently the optimizer added noise to the samples, but no patterns emerged. Figure 5.6 shows an example of difficulties in finding an optimal input by demonstating an input belonging to normal and the same sample after optimalization belonging to AF.

Convolutional kernels. Thirdly, we gave a trial to plot the kernels of the convolutional layers. These does not yield either any result, no recognizable pattern were present on the plots of kernel graphs.

Conclusion. The failure of these experiments mostly might be the effect of the mixed time- and frequency domain feature extraction. While that multi-domain processing method improved out final score, it is a huge drawback when propagating the information backward in the model because it is hard to think in frequency domain for people, and people hardly can find any frequency domain features (even if it is obviously there) on a time domain representation.

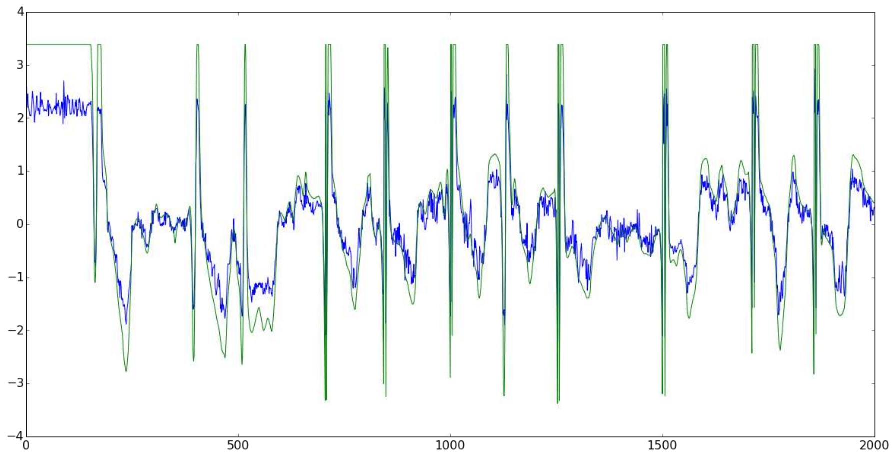


Figure 5.6: An example of difficulties in finding an optimal input. The green line is the original input classified by the model confidently to the class AF. On the other hand, the blue line shows the same sample after optimizing for normal class. That optimized sample would be classified firmly as normal, even tough a human could tell immediately that the the recordings are the same.

6. Summary

Contributions

In the previous chapters we described our main contributions to the field of signal processing and atrial fibrillation detection. We worked out the major cornerstones of designing deep neural networks specifically for cardiac monitoring while keeping complexity in mind. We consider our biggest achievement that using computer vision experience we constructed an algorithm, trained it from scratch on a moderate sized data set with high error in the ground truth labels that outperforms the referenced state of the art cardiac failure detectors. In this chapter we also mention the final scores of the *CinC Challenge of 2017*, and review their algorithms and inspect ways to improve ours with such knowledge.

Challenge results and collaboration

Our results. While the performance of the classifier can be measured many different ways, we have decided to stick to the official scoring algorithm, that is, the F1 score. Our final model was a parallel ensemble of a feature extractor on temporal domain (EncodeNet) and on spectral domain (skipFCN) that yielded the following results:

- Detailed F1 on training set: 0.9093 (normal), 0.8883 (AF), 0.8181 (other), 0.5589 (noise)
- Overall F1 on training set: 0.87
- Detailed F1 on test set: 0.8786 (normal), 0.7953 (AF), 0.6887 (other), 0.6404 (noise)
- Overall F1 on test set: 0.79

Besides that we contacted two cardiologists from the Heart and Vascular Center of Semmelweis University. By their help, we can state that the model detects some of the most important features doctors are watching when making a diagnosis and in many cases it makes similar decisions as doctors do (e.g. the recordings that are hard to classify for the model are difficult for doctors as well).

Best competitors. The challenge was really tight and the top results clearly represents the level of difficulty. The following competitors achieved a test score of F1=0.83

- **Zabihi et al.** Detection of Atrial Fibrillation in ECG Hand-held Devices Using a Random Forest Classifier [83]. With heavy feature-engineering provided over 500 hand-crafted features fed to a traditional machine learning approach, random forests. These representations of the samples could be easily concatenated to our multi-domain feature vector - also experimenting with random forests could yield some interesting results.
- **Teijeiro et al.** Arrhythmia Classification from the Abductive Interpretation of Short Single-lead ECG Records [73]. Using an ensemble of 3 Long Short Term Memory networks and an MLP stacked on the top, with additional expert feature support to correct samples that yields low confidence values. They describe a method of training of parallel recurrent modules that could help us elaborate on our experiments with LSTMs.
- **Datta et al.** A Robust AF Classifier using Time and Frequency Features from Single Lead ECG Signal (not published yet). Basically their approach is closely related to ours, collaboration would reveal how could we improve our proposed multi-domain networks.
- **Hong et al.** ENCASE: an ENsemble ClASsifiEr for ECG Classification Using Expert Features and Deep Neural Networks (not published yet). They have fine-tuned a 64 layered Residual Neural Network supported by experts who relabeled the training set and removed uncertain samples. Their efforts and capacity could help us train a version of our classifier of higher complexity.

Future work

Short-term plans. After successfully training our final model, we intend to use the convolutional layers as feature extractors in reverse engineering to find out if we can help cardiologists by providing them a list of ECG patterns our network used as a guideline for classification. For doing so we have multiple choices: we can utilize DeConv nets [84], or apply gradient ascension [81] on the receptive field of perceptrons, or simply take the mean of windows in samples that yields the largest activation in the latent feature representation.

From the feedback of the cardiologist, we learned that we could improve our program by increasing the sensitivity of AF because it is much better to classify a normal as AF than the opposite. Furthermore, implementing an algorithm that detects the quantity of irregularity might improve the performance.

Long-term plans. We would like to implement Domain Adversarial training of Neural Networks (DANN) [16] to transfer our best model's trained parameters to real world applications which may introduce different sample density.

Our following task will be utilizing One Shot learning [67, 76] to tackle the small-train set problem.

Finally, the device that was used to record the original samples will be commercially available soon, and it is a great opportunity for us to write the inner mechanism of a real life, end-to-end product.

Appendices

A. The cardiac conduction and stimulus formation

The heart conduction system controls the generation and propagation of electrical signals or action potentials. They cause heart muscles to contract and the heart to pump blood.This electrical activity can be measured by electrodes placed at specific points on the skin through the recording knows as Electrocardiogram (ECG). A tracing of the overall electrical activity of the heart is possible, resulting from the propagation of many action potentials.

In the normal heart each beat begins in the right atrium with an action potential signal from a sinal atria (SA) node, the heart's natural pacemaker. The signal spreads across both areas, because the muscle cell depolarizes in contraction. This induces the face known as Atrial Systole. On the ECG this atrial depolarization is represented by the P wave (Fig. A.2).

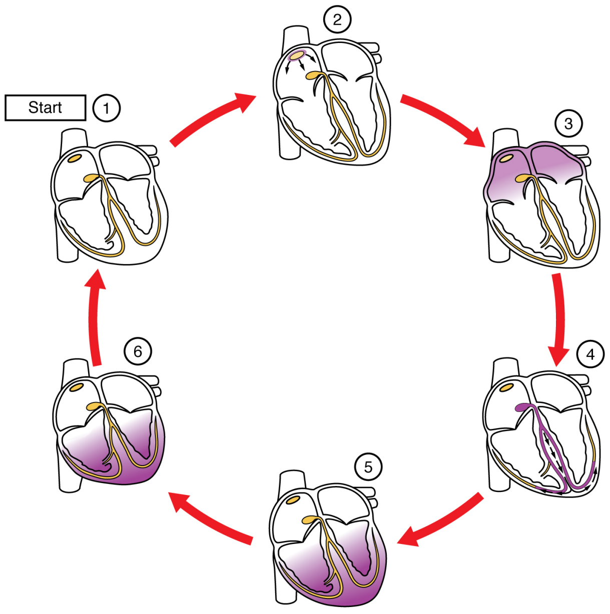


Figure A.1: The cardiac conduction and stimulus formation

The period of conduction that follows atrial systole and precedes the contraction of the ventricle is depicted on the ECG by the PR Segment, a flat line following the P wave. When the signal leaves the atriait (Fig. A.1 Phase 3) enters with ventricles via the atrial ventricular (AV) node located in inter atrial septum (Fig. A.1 Phase 4). It enters the Bundle of His and spreads through the Bundle branches and the large diameter Purkinje fibres along the ventricle walls (Fig. A.1 Phase 5-6).

As the signals spread through to the ventricles the contractile fibres depolarize and contract very rapidly, inducing Ventricular systole. The ECG QRS Complex (Fig. A.2) represents this rapid ventricle depolarization. Atrial depolarization also occurs in this time. But any atrial activity is hidden on the ECG by the QRS complex. Finally as the signal passes out of the ventricle, the ventricular walls start to relax and recover in the state described as Ventricular diastole. The T wave (Fig. A.2) on the ECG marks this ventricle re-polarization.

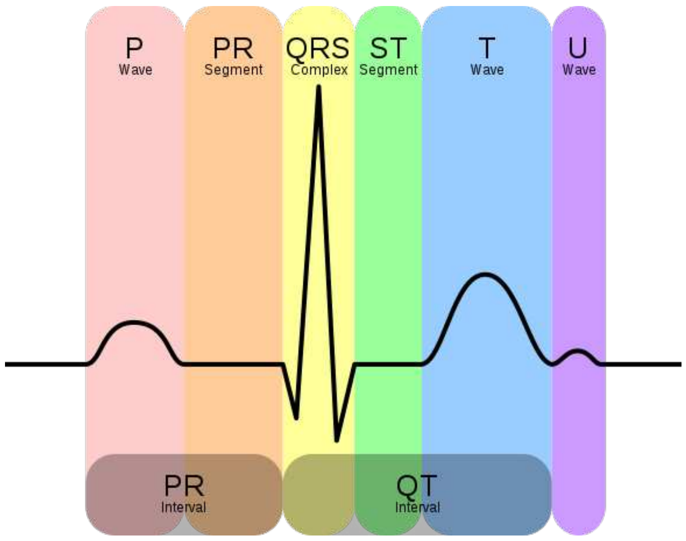


Figure A.2: Schematic figure of ECG curve. Image Credits: Hank van Helvete - EKG Komplex

On the ECG the ST Segment (Fig. A.2) depicts the period when the ventricles are depolarized. QT interval represents a summation of all ventricular APs, LQTS mutations primarily affect ventricular ion channel complexes. The sequence of events just described and associated ECG traces repeat in every heartbeat. An ECG is not a tracing of single action potential, but rather the amalgamation of many action potentials, which constitute the electrical activity of the heart.

B. Most and Least Confident Classification Cases

Most confident classification cases of normal class

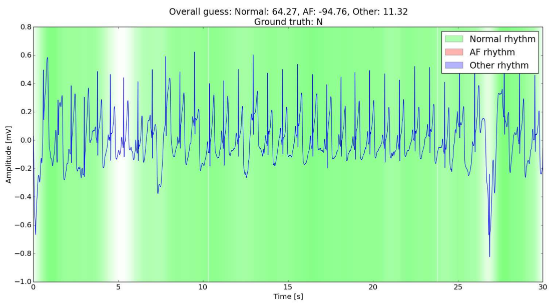


Figure B.1

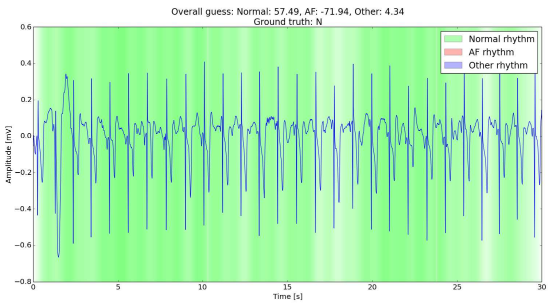


Figure B.2

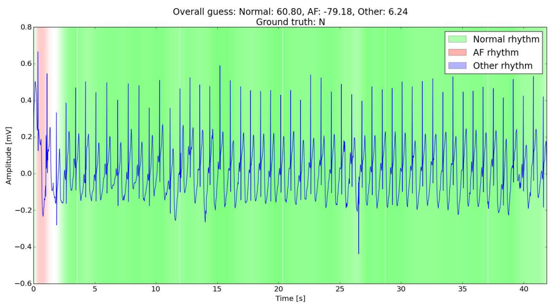


Figure B.3

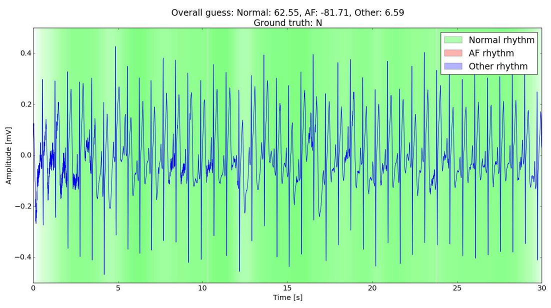


Figure B.4

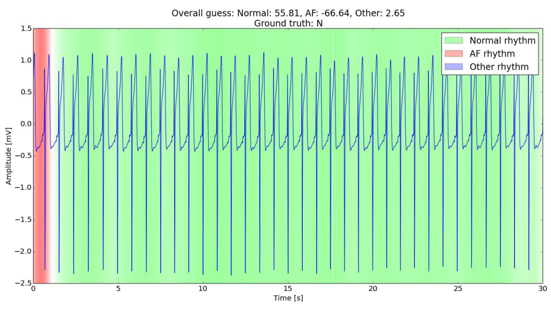


Figure B.5

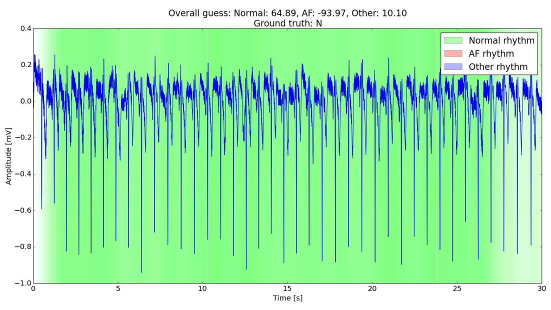


Figure B.6

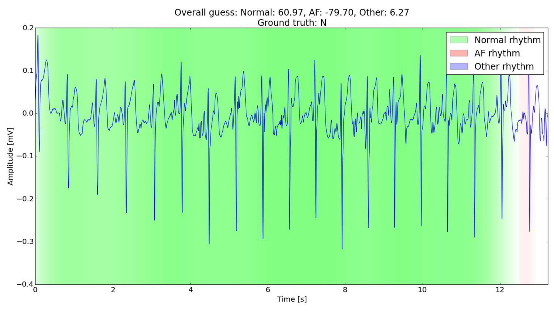


Figure B.7

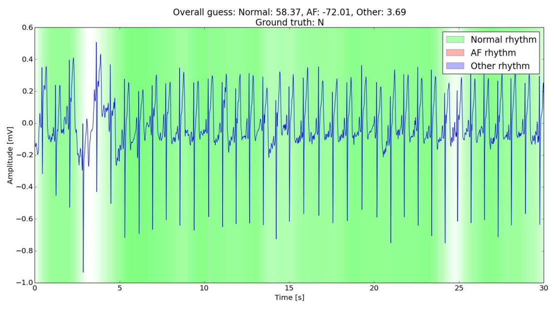


Figure B.8

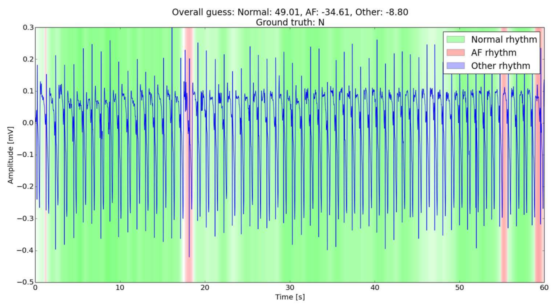


Figure B.9

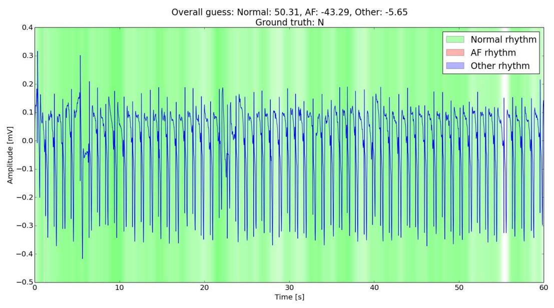


Figure B.10

Most confident classification cases of AF class

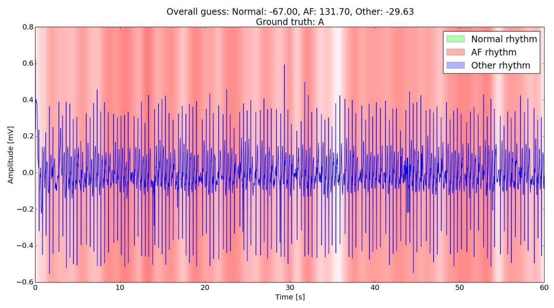


Figure B.11

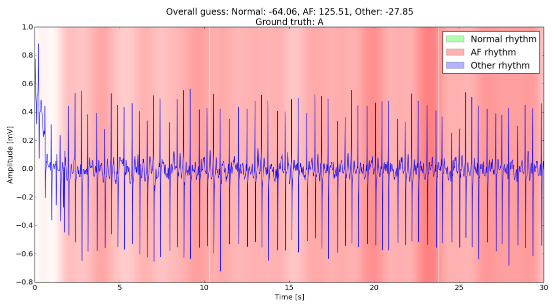


Figure B.12

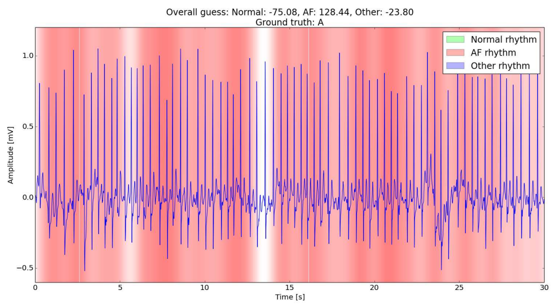


Figure B.13

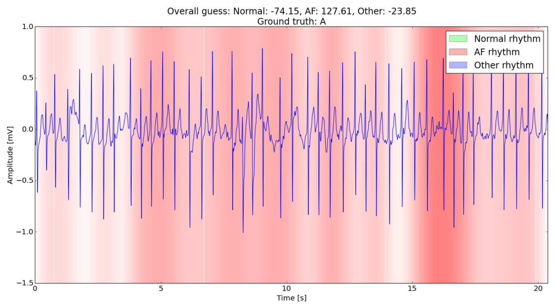


Figure B.14

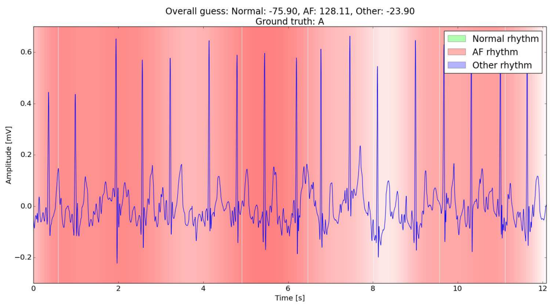


Figure B.15

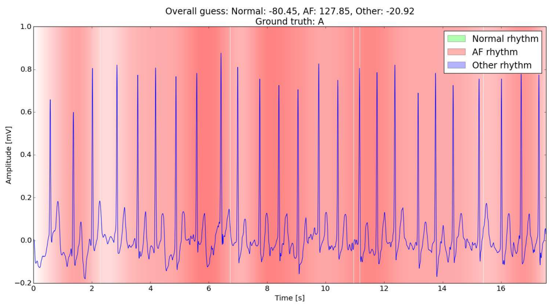


Figure B.16

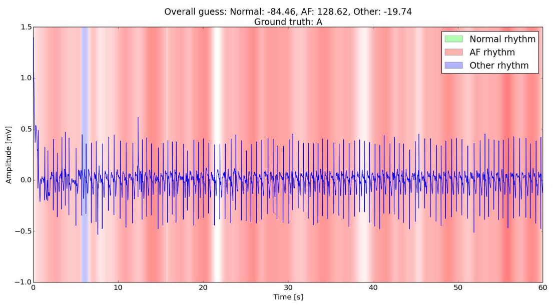


Figure B.17

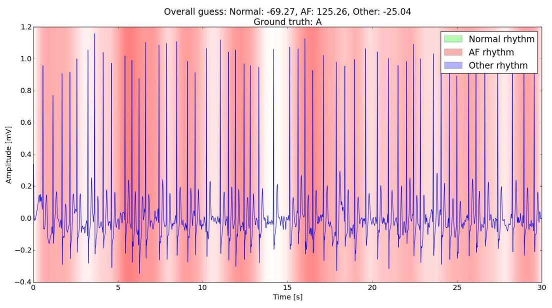


Figure B.18

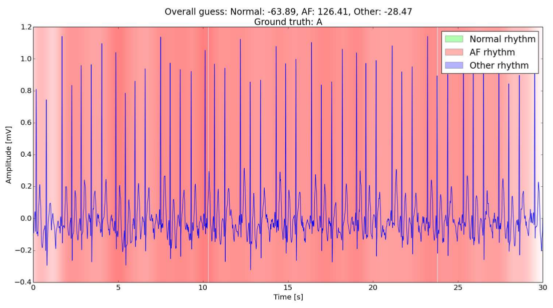


Figure B.19

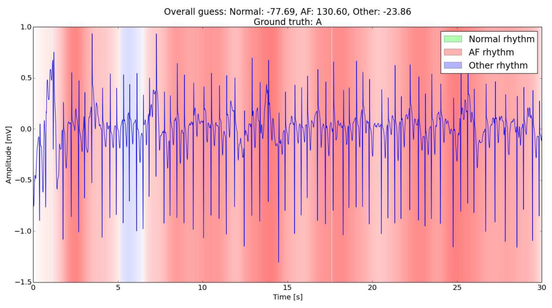


Figure B.20

Least confident classification cases of normal class

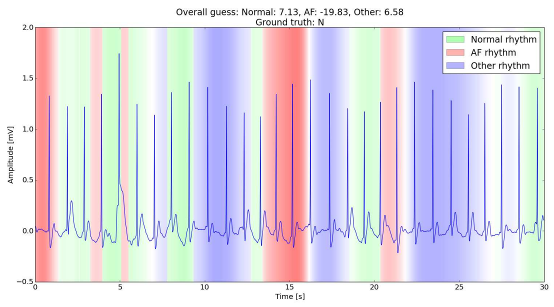


Figure B.21

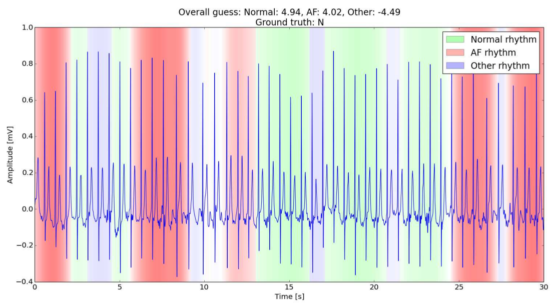


Figure B.22

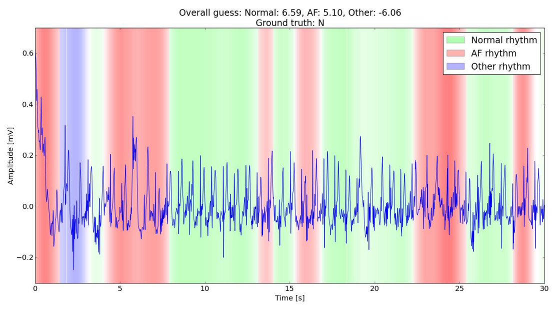


Figure B.23

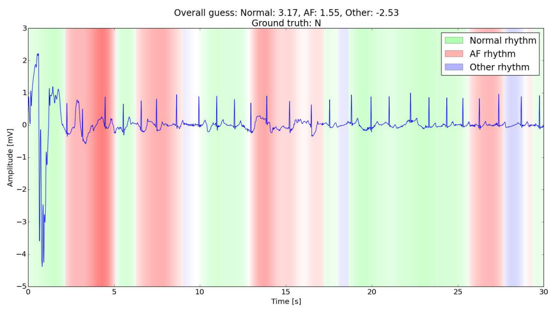


Figure B.24

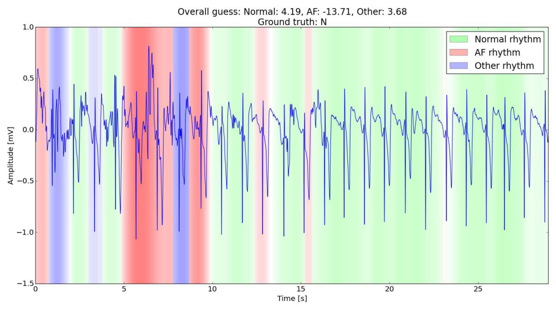


Figure B.25

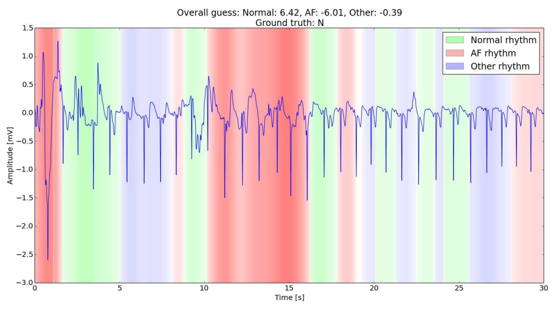


Figure B.26

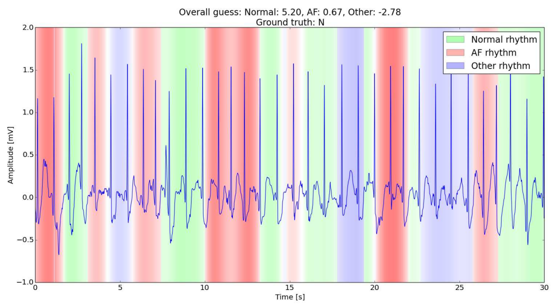


Figure B.27

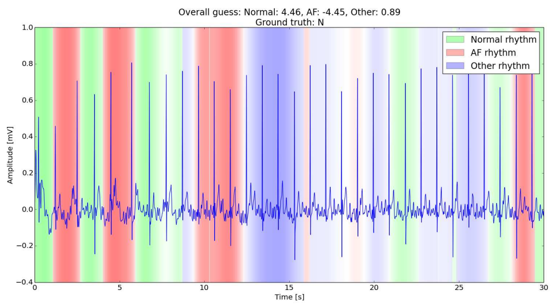


Figure B.28

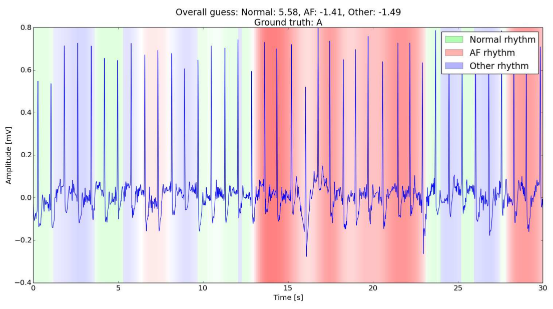


Figure B.29

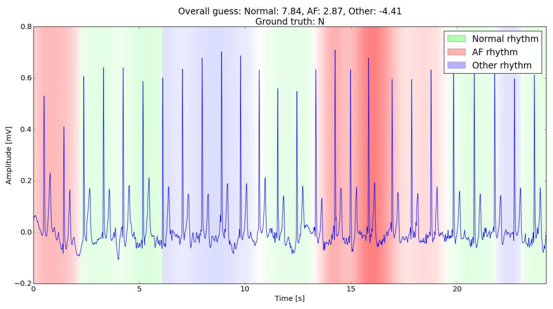


Figure B.30

Least confident classification cases of AF class

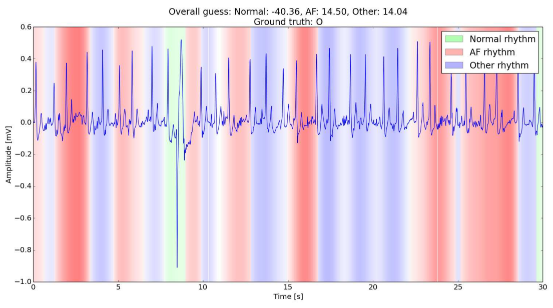


Figure B.31

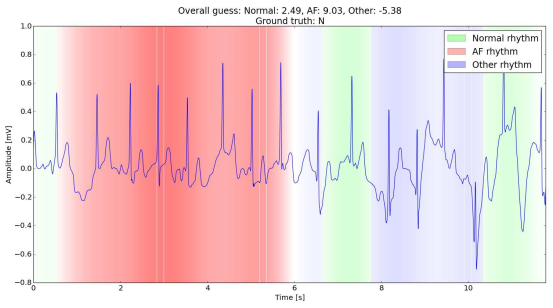


Figure B.32

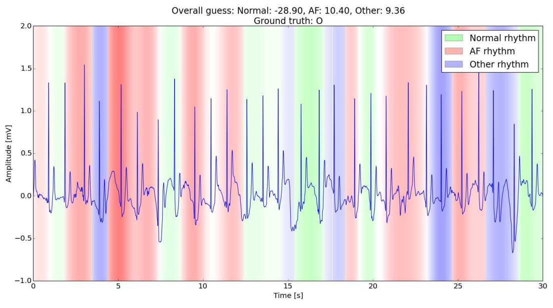


Figure B.33

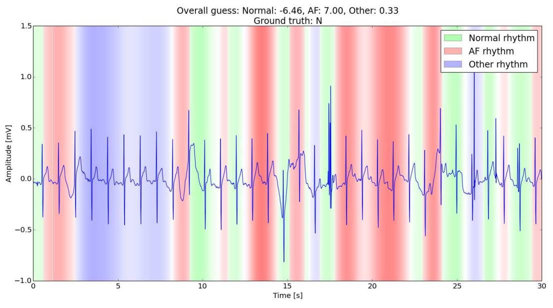


Figure B.34

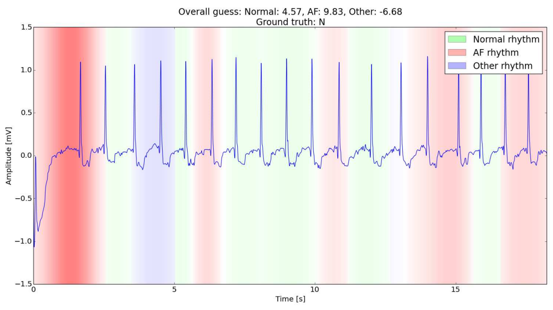


Figure B.35

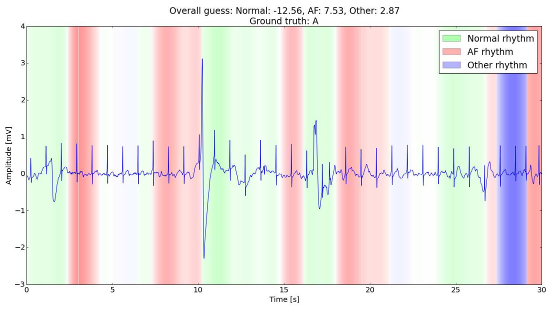


Figure B.36

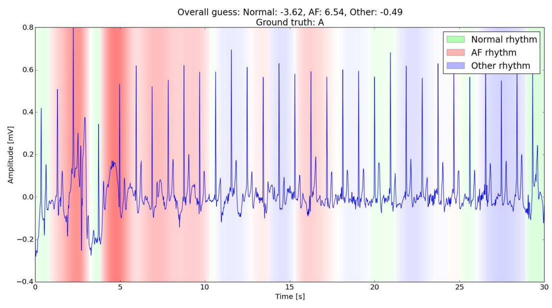


Figure B.37

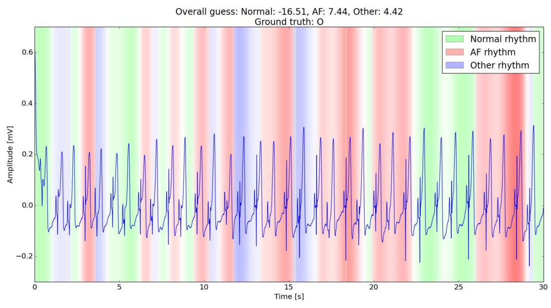


Figure B.38

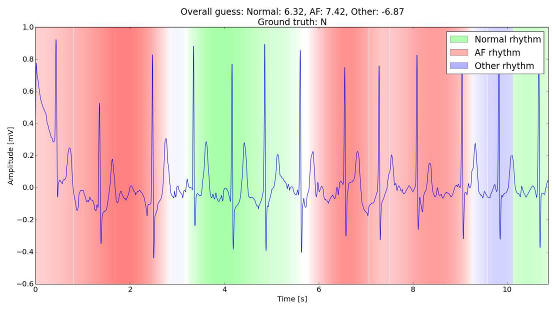


Figure B.39

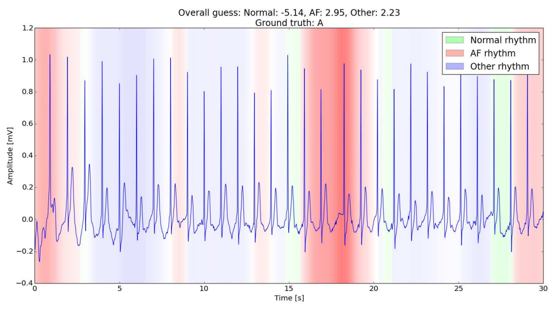


Figure B.40

C. Neural Network Basics

Single Layered networks

For example if a small picture is given to a person, he or she guesses about it. If that person is told that there are four classes, like *Car, Plane, Cat, Kid*, he can tell how likely it is that the given picture falls in that class, so he can give a so-called *confidence parameter*. Patterns which are associated with cats may be associated with kids too, but is unlikely to be associated with planes. Deciding whether a pattern improves the confidence on each class or not, yields a **sign** for the given pattern, and the measure how strongly it influences the likelihood is called the **weight** of the pattern. In *neural networks* there are small nodes based on the biological model of neurons, that is responsible for recognizing and weighting the patterns. These nodes are called **perceptrons**.

Defining the Perceptron

Let x be preprocessed information, the perceptron $\mathbf{P}$ decides which parts of it are important in recognizing a cat: having a corresponding weight with large magnitude, and which of them are irrelevant: having a weight with magnitude close to zero. Therefore a *perceptron* has a weight w_i for each input x_i and its transient state can be now formally written:

$$\mathbf{P}_{transient} = \sum_i x_i w_i$$

Which can be also rewritten as the inner product of $\mathbf{x}$ and $\mathbf{w}$

$$\mathbf{P}_{transient} =< \mathbf{x}\ ,\ \mathbf{w} >= \mathbf{x}^T\mathbf{w}$$

In the following examples perceptrons will be arranged in a special way, to form a **Feed Forward** network, meaning that the information flow will occur in a direct way, without feedback - see figure C.1. However wiring many perceptrons together results in a numerically unstable system, caused by the lack of any restriction on the magnitude of the weights. Regularization addresses exactly this problem, implicitly making the network less likely to simply memorize pictures, or only respond to samples from the training set – the problem of overfitting.

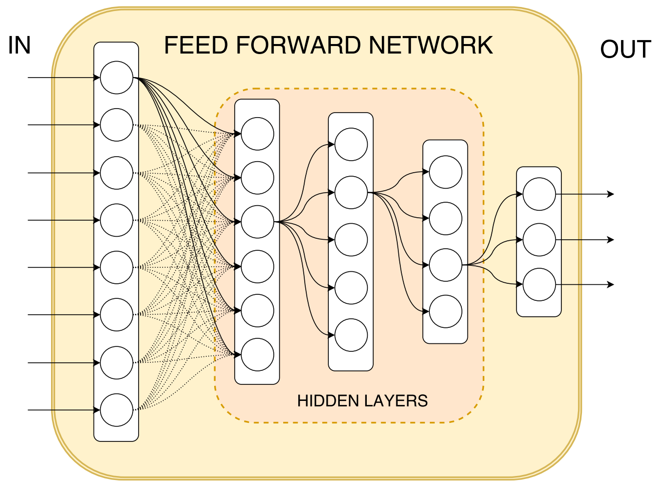


Figure C.1: A small feed forward network with three hidden layer composed of Fully Connected layers.

Basics of Learning

After describing the model, the first question is:

— *what are the correct values for* $\mathbf{w_P}$ *for each* $\mathbf{P}$ *in network* $\mathcal{F}$*?*

There are a lot of intuitive explanations how this problem should be approached and solved. For an exhaustive study of the topic, visit Michael Nielsen's website [53]. Anyhow, in general this is still the most studied question in machine learning. Luckily, if $\mathcal{F}$'s performance can be measured, thanks to its feed-forward structure a *numerical suggestion* can be defined as well, which tells how to change $\mathbf{w_P}$ to improve efficiency of $\mathcal{F}$. In practice an $\mathcal{L}$ Loss function is defined, and the objective of the training is to reduce it. Without digging deep in math we can find an intuitive situation with the same results.

Take a perceptron $\mathbf{P}$ with input $\mathbf{x}$ of objects on a given picture of a cat. Suppose that this perceptron *fires* (has high output value) when wheels are on the picture. It should remain silent when there are no round objects listed in $\mathbf{x}$, still it turns on, affecting the output of $\mathcal{F}$, producing high error. Since we know the original label of the picture, we can tell which assumptions were wrong, and which were good - which to decrease, which to increase if the next time $\mathcal{F}$ is given the same input. This information is distributed between the previous perceptrons which caused the current one to fail, by simply multiplying the error with the weight corresponding to the previous node. Also with the information of how much the output of the actual node influenced the error, and with its input values, the significance of misleading $\mathbf{x_i}$ can be reduced, and important features' weights can be increased, which is actually finding a better $\mathbf{w_P}$. For further intuitive examples see section 5, or visit Andrej Karpathy's guide [31].

SGD. The example led to the practical application of the so called *Stochastic Gradient Descent*. Originally every samples in the training set should be introduced to the system before updating its weights and that would be the *Batch Gradient Descent*, but in the hope that the samples fall in a subset of the whole space of all possible inputs, the algorithm uses mini-batches for the learning process. When the number of samples in the batches reduces to 1, the training is called on-line training. And if it happens on $\mathcal{F}$ containing a single perceptron it is called *Rosenblatt perceptron learning algorithm.*

Inference

Let us suppose that for a particular task an $\mathcal{F}$ is given, first what we have to understand is how input data $x \in \mathbb{X}$ (interchangeably x_1) is inferred. First, assume that the data can be expressed as multi-dimensional matrix, like RGB pictures, audio recordings, gene maps. Some networks preserve the spatial information i.e. feature extraction performed on images, while other instances operate on the whole input data i.e. processing audio samples in frequency domain. Either way the output y (or y_L) can be obtained by feeding x to the network: $y = \mathcal{F}(x)$ The core concept is that we can compose such an $\mathcal{F}$ function by applying multiple projections to x. Practically that means sending input through the first layer, the second, and all the way through to the last layer L^{th}, which output would be the value of $\mathcal{F}(x)$, the response of the network. Therefore in terms of evaluating $\mathcal{F}(x)$ layer by layer, actually translates to a single function call, which can be unfolded to a sequence of embedded projections:

$$\mathcal{F}(x) = F_L\left(x_L\right) = F_L\left(y_{(L-1)}\right)$$

$$F_L\left(y_{(L-1)}\right) = F_L\left(F_{(L-1)}\left(x_{(L-1)}\right)\right) = F_L\left(F_{(L-1)}\left(\cdots F_1(x)\right)\right)$$

Using the function composition operator $\circ$, rewritten in the classical notation:

$$\mathcal{F}(x) = F_L \circ F_{(L-1)} \circ \cdots \circ F_1(x) \tag{C.1}$$

The C.1 equation is the most fundamental idea behind feed-forward neural networks, namely the *inference* or *forward-propagation* As mentioned above, every layer l is represented by an F_l. The most basic layers are the *Fully Connected* and *Activation* layers.

Fully Connected layer

These layers carry out the heavy-lifting of inference by performing linear projection and translation transformations. The operations are following the rules of basic linear algebra, where the input $x_{FC} \in \mathbb{R}^N$ and the output $y_{FC} \in \mathbb{R}^M$ are specified as real valued vectors. The parameters of the layer $\phi_{FC} = (W, b)$ are the corresponding weights and biases of each perceptron node in the layer forming a *weight matrix* $W \in \mathbb{R}^{M\times N}$ and a *bias vector* $b \in \mathbb{R}^M$ respectively. Therefore evaluating the output of the Fully Connected layer is the defined by the following:

$$\begin{aligned} y_i &= \left(\sum_j W_{i,j}\; x_j \right) + b_i \\ y &= W \cdot x_j + b \\ [M] &= [M \times N] \cdot [N] + [M] \end{aligned} \tag{C.2}$$

Activation layer

Nodes in activation layers are introducing non-linearity to the network, by applying the same non-linear activation function to the corresponding output of the previous layer, performing element-wise operation. Let F be an activation layer with activation function f after a fully connected layer with 3 neurons:

$$F(x) = \begin{pmatrix} f(x_1) \\ f(x_2) \\ f(x_3) \end{pmatrix}$$

These functions are essential for the network, since they increase the numerical stability: they *squeeze* or *mitigate* the input preventing the network from *saturation* or *explosion* (numerical of course). Conventionally the following functions are applied most often as activation function:

$$\text{RectifiedLinearUnit(ReLU)} := max\,\{0, x\} \tag{C.3}$$

$$\text{HyperbolicTangent(TanH)} := \tanh(x) = \frac{2}{1+e^{-2x}} - 1 \tag{C.4}$$

$$\text{SoftPlus(SP)} := \ln(1+e^x) \tag{C.5}$$

$$\text{Logistic(Log)} := \frac{1}{1+e^{-x}} \tag{C.6}$$

The only constraint on these functions that they have to keep the dimension of the input, namely $F_{act} : \mathbb{R}^N \mapsto \mathbb{R}^N$. *Note:* These functions do not have any variable parameters, therefore activation layers cannot be trained.

Measuring efficiency

If the function $\mathcal{F}$ mentioned above is given, and satisfies our needs, then we are done. However this is usually not the case, and finding the optimal $\mathcal{F}^*$ is the main challenge targeted by many branches of Machine Learning. Despite it was proven, that standard multilayer feed-forward networks are capable of approximating any measurable function to any desired degree of accuracy [25], if the goal function $\mathcal{F}^*$ is unknown, or too abstract to be *measurable* (i.e. telling how funny a picture is), we cannot utilize the universal approximator.

Loss

By reformulating the objective, we can define a Loss function $\mathcal{L} : \mathcal{F}(\mathbb{X}) \mapsto \mathbb{R}$, which maps our candidate $\mathcal{F}$ network to a scalar field, that represents the general correctness of $\mathcal{F}$ over the space of possible inputs $\mathbb{X}$ – the lower its value the better $\mathcal{F}$ is performing. By doing so we may apply Machine Learning algorithms that would *minimize* the Loss, therefore $\mathcal{F}$ would converge towards $\mathcal{F}^*$ implicitly.

Note: In practical implementations $\mathcal{F}$ is not evaluated over the whole space of possible inputs, instead in the hope that a small subset of both *training* and *validating* samples called a mini-batch will approximate $\mathcal{L}(\mathcal{F})$ as well. Useful practices for reducing computing complexity and improving stability, and rate of convergence will be covered later.

Supervised learning

In cases where the parameters of $\mathcal{F}^*$ are not known, but we know how it would map the input space $\mathbb{X} \mapsto \mathbb{Y}$, e.g. which character appears on the input image x, then we can define a set of previously *labeled* pairs of input - solution sample which could be used later on for training, and evaluating performance of the network.

Unsupervised learning

When no labeled dataset is available, the network still can be used for extraction of hidden structure of the unlabeled samples. Later these instances are used as density estimators, or adapted as feature extractors for larger networks. **Generative Networks.** Training such architectures can be done by feeding networks random noise as input and training them to reproduce given samples: $\mathcal{F} : \mathbb{R}^k \mapsto \mathbb{X}$, hence the name *Generative Networks.*

Auto-encoder Networks The other frequently applied paradigm is setting the objective task to compress the input sample into a $h \in \mathbb{R}^k$ hidden representation vector that is able to preserve the key information about the original input, and either by symmetric (Restrictive Boltzmann Machines) or independent (Deep Belief networks) operations decompress the data.

In both cases hyper-parameter k is intuitively the number of unlabeled features in the *latent space* that the network will be able to categorize, i.e. correlation between different color intensities on pictures taken of stained brain samples.

Adjusting parameters

Generally speaking, applying Machine Learning algorithms boils down to the process of iteratively altering parameters ϕ of $\mathcal{F}$ to optimize the Loss. Once $\mathcal{L}(\mathcal{F})$ is obtained, we can evaluate how changing the parameters would influence it – evaluate the gradient $\nabla_\phi \mathcal{F}$ of the parameters with regards to $\mathcal{L}$.

Gradient Descent

Updating ϕ by descending on the gradient slope with a small step size ϵ will decrease $\mathcal{L}$. Derivation from the general form of Gradient Descent depends on the architecture of the network, but there is a main concept for doing so, called Backpropagation described by Werbos *et al.* [80]. The exact method which can be applied for Fully Connected networks will be explained in detail in section 5.

Training Policies

Though theoretically after finite steps of iterations, with Gradient Descent $\mathcal{F}$ may approximate any Borel-measurable function, in practice Deep Neural Networks can fall into local-minimum of the parameter-space. I.e. the network tends to learn the very basic features of the input space, if it is not forced to generalize. The problem with generalization is that the training set has finite samples and extending it requires human-supervision. Therefore training networks with many layers requires some advanced techniques for training. There are multiple way to improve the *convergence rate* and the *stability* of the network.

Batch processing. There are two radical methods of updating the parameters in the network.

Theoretically if we wanted to create a perfect network, we would infer all possible samples from $\mathbb{X}$ and the Loss function $\mathcal{L}$ would evaluate the network on every solution (in a single iteration, without the network being updated), then we would take the mean of the Loss to evaluate the gradient, e.g.

$$\mathcal{L}^* = \frac{1}{N} \sum_{x \in \mathbb{X}} \mathcal{L}(\mathcal{F}(x))$$

One update on the whole training set is called a **batch** This way, if the capacity of $\mathcal{F}$ is large enough, it is possible to train the network to be able to solve every problem. In a special case when we want to simulate logical circuits with real values instead of Boolean `true` or `false`, then we can train a network to act like a logical processing unit - since we are able to map $\forall x \in \mathbb{X} \mapsto y \in \mathbb{Y}$ by external evaluation.

The other case is more practical, when the train data is not available at once, but in sequence. The network is usually updated in *testing* time and that is why it is called **on-line** training. The topic is described exhaustively in [66]. In practice the network is trained previously before on-line training (even if a very few sample are available), because the responses are not just used for evaluating the Loss function, but taken into account as valid predictions. Normally there is no opportunity to label freshly acquired samples, and this is the main reason that this technique is used mainly in *unsupervised learning* tasks. However when supervision is available, then the network can be trained in the following manner: predictions which were correct are put into the training set, and the network weights are updated immediately to encourage the same response for similar samples. A frequent application of on-line training is called *Q-learn*, for case studies see [65].

Both methods are powerful for specific tasks, but in practice the solution is in between: **mini-batch** training. Inferring multiple samples at the same time with the same network not just yields a more stable convergence, but is an excellent opportunity to parallelize the process: Extending by one extra dimension to each stage of the inference (inputs, transient activations, outputs) can be done easily, also almost every library can take the advantage to perform optimized matrix operations on tensors (at least *NumPy* can). On the second hand, processing only *some* instead of all of the input samples at once is computationally a better choice. In real applications it is worth considering 2^k mini-batch sizes because of the requirement of memory allocation [32].

Advanced First order derivatives. Simple heuristics applied to first order derivatives, such as:

Momentum [71] $v_{(n+1)} = \gamma v_n + \nabla_{\phi_n}\mathcal{L}$ and $\phi_{(n+1)} = \phi_n - \epsilon v_n$

Adagrad [15] $r_{(n+1)} = r_n + (\nabla_{\phi_n}\mathcal{L})^2$ and $\phi_{(n+1)} = \phi_n - \frac{\epsilon}{\gamma+\sqrt{r_n}}(\nabla_{\phi_n}\mathcal{L})^2$ (element-wise)

RMSProp [74] $r_{(n+1)} = (1-\epsilon)r_n + \epsilon(\nabla_{\phi_n}\mathcal{L})^2$

Adam [34] Combination of RMSProp and momentum.

A more sophisticated, but expensive method is using second order derivative, for instance Conjugate gradient [69].

Decreasing the learning rate ϵ. For large networks it is inevitable to use decreasing learning rate, otherwise the convergence would take too much time (small ϵ) or would not even occur (large ϵ).

Cross-Validation. Special $\hat{\mathcal{L}} = (\mathcal{L}_{train}, \mathcal{L}_{validation})$ functions approximate the efficiency of the network not just by evaluating them on the currently or previously trained samples, but on samples which are from a totally **disjunct set** $\mathbb{X}_{validation}$. The motivation behind it is to see how well would the network perform on samples which it has not faced before. By doing so we can evaluate how well the network *generalized* the information from samples of the training set. In order to keep $\mathbb{X}_{train} \bigcap \mathbb{X}_{validation} = \varnothing$ we may not use $\mathcal{L}_{validation}$ for parameter updating. However in order to **prevent overfitting** we can monitor e.g. generalization error $E_g = \frac{\hat{\mathcal{L}}_1}{\hat{\mathcal{L}}_2}$ during training and use **Early Stopping** when $E_g > \delta$, where δ is a parameter of how much we tolerate overfitting.

Networks in practice

Though networks can vary in shape and function, the representation of the succeeding layers, evaluated by embedded functions described in C.1 is a common feature. Each of these layers serve as nodes of the computational graph of the network. The type of the layer determines whether it can be updated or not: for *Fully Connected* layers $\nabla_\phi F_{FC}$ is well defined, explained in the following example.

Toy example. Assume $\mathcal{L}$ is given, and we have a fully connected network with 2 hidden layers, i.e. $L = 3$, which maps a $\mathcal{F}$ dimensional input vector x to a M dimensional output vector y, namely $\mathcal{F} : \mathbb{R}^N \mapsto \mathbb{R}^M$. The constraint on the parameters are:

W^1 of the first layer must have N columns

W^3 and the bias b^3 of the last layer must have M rows, dimensions respectively.

Number of rows of W^l must match $dim(b^l)$ 3 dimensional

Number of columns of W^l must match $dim(b^{(l-1)})$

Then the evaluation unfolded would look like the following:

$$y = \mathcal{F}(x) = F_3 \circ F_2 \circ F_1(x) = W^3(W^2(W^1(x) + b^1) + b^2) + b^3$$

For further usage and simplicity, I would like to fix these numbers. Let $\mathcal{F}$ be a network with the following *shape*: $[5, 4, 3]$ meaning that in each layer there are 5, 4, 3 neurons respectively, $M = 3$ and all nodes are connected to the previous layer. The width of the input layer is not yet defined, let it be $N = 10$. *Note:* It is usually distracting and redundant to explicitly write the width of the outermost layers when testing different networks, because the input and the output layers must have fixed dimension for the same task, while the width of the hidden layers are varied. Define an L_2 Loss function on the toy example. For one sample-label pair (x, y^*) the L_2 Loss is:

$$\mathcal{L}_2(\mathcal{F}) = \frac{1}{2}\sum_i (y_i^* - \mathcal{F}(x)_i)^2 = \frac{1}{2}\sum_i (y_i^* - y_i)^2 \tag{C.7}$$

L_2 is a universal Loss function, which is used in cases where the label space $\mathbb{Y}$ is continuous (e.g. floorspace, consumption and height of a house, based on the price: $\mathbb{R}^1 \mapsto \mathbb{R}^3$).

Differentiation

In the above evaluating of $\nabla_\phi \mathcal{F}$ can be done in two ways, namely by numerical approximation, or by analytical derivation, in the following I will discuss both.

Numeric differentiation

Evaluating the numerical gradient (or difference) is an elementary, yet powerful operation, in which we would *perturb*, or modify one parameter ϕ of our system $\mathcal{F}$ at once. That is done by first adding ϕ^+ and after subtracting ϕ^- a little amount $d\phi$ from the original ϕ and evaluate $\mathcal{L}^\pm = \mathcal{L}(\mathcal{F}_{\phi^\pm})$, namely the *Loss* of the system in the modified state, yielding the numerical gradient in the following equation:

$$\frac{d\mathcal{L}(\mathcal{F})}{d\phi} = \frac{\mathcal{L}^+ - \mathcal{L}^-}{2d\phi} = \frac{\mathcal{L}(\mathcal{F}_{\phi^+}) - \mathcal{L}(\mathcal{F}_{\phi^-})}{2d\phi} \tag{C.8}$$

Summary. In a nutshell value of $\frac{d\mathcal{L}(\mathcal{F})}{d\phi}$ tells how changing the parameter ϕ by $d\phi$ would change the performance of the network. If it is positive then updating $\mathcal{F}$ by adding $d\phi$ to ϕ would result in higher Loss value, which is the opposite of our goal, so we just subtract it, if it is negative, then trivially we should add $d\phi$ to ϕ since it is making some good progress.

Complexity

Though letting the computer do the hard work seems to be a good idea, it is worth considering that the simple method above will be applied to every ϕ of $\mathcal{L}$. It means that for the network in the toy example, we need to evaluate $\mathcal{L}(\mathcal{F})$ two times for each parameter in the weight matrices and the bias vectors of the network, totaling in

$$\#(\phi) = 2 \times \sum_{i=1}^{3} N_{(i-1)} \cdot N_i + N_i = 2 \times (10 \cdot 5 + 5 ... + 4 \cdot 3 + 3) = 188$$

Even if $\mathcal{F}$ is approximated by using k-sized mini-batches for evaluation it is still a computationally very expensive function, because the inference would result in the following number of operations of addition and multiplication:

$$\#(\text{operations}) = k \times \sum_{i=1}^{3} 2 \cdot N_{(i-1)} \cdot N_i + N_i = k \times 176$$

Therefore approximated with $k = 10$ mini-batches would a single parameter update of a very tiny network would require total operations of:

$$\#(\text{total}) = \#(\phi) \times k \times \#(\text{operations}) = 188 \cdot 10 \cdot 176 = 330880 \tag{C.9}$$

Because both $\#(\phi)$ and $\#(\text{operations})$ has complexity of $\mathcal{O}(N^2)$, one update will yield complexity of

$$\#(\text{total}) = \mathcal{O}(N^4) \tag{C.10}$$

We can see that even for a shallow and relatively small network (industrial Awe networks has billions of parameters, and uses much larger batches) described in the toy example the method is really costly. That encourages us to derive our differentials on paper first, and use *numerical gradient approximation* for checking our solution. Using *Gradient Check* is essential when implementing new architectures, because it is a very efficient tool for debugging in comparison with updating. The method in a few words is about setting an error rate ϵ, and decrease $d\phi$ until the numeric solution does not match the analytic solution with $1-\epsilon$ significance. If the analytic solution is incorrect the cycle will not terminate.

Analytic differentiation

Deriving the update by hand requires basic knowledge in calculus extended to multivariate cases, though since the operations are elementary, in general we must understand only three basic definitions to do so. Some formality before starting: we have three independent variables x, y, z and functions f, g, h. The result of operations performed on variables, i.e. $x + 2y$ can be represented by a function $f = x + 2y$. If the value depends on a variable then it can be written explicitly, passing the variable as *the argument* of the function $f(x, y) = x + 2y$. For the sake of simplicity assume that the variables are not general objects from an abstract space, they are only real values: $x, y, z \in \mathbb{R}$. However the following description could be extended for the above-mentioned variables as well. For any one-dimensional function $f(x) : \mathbb{R} \mapsto \mathbb{R}$ we say that the value represented by the function depends on the variable by the extent of its derivative. The derivative (or differential) of the function can be seen as an ideal case of C.8 where the perturbation would approach zero, namely:

$$\frac{\partial f}{\partial x} = \lim_{dx\to 0} \frac{df(x)}{dx} = \lim_{dx\to 0} \frac{f(x+dx) - f(x-dx)}{2dx} \tag{C.11}$$

Multiplication rule. Consider a value $x \cdot y \cdot z$ represented by $f(x, y, z)$. f is now depending on three variables, we can define the measure of this dependency on one variable by the formal equation:

$$\begin{aligned}\frac{\partial f}{\partial x} = \lim_{dx\to 0} \frac{f(x+dx,y,z) - f(x-dx,y,z)}{dx} &= \lim_{dx\to 0} \frac{((x+dx)\cdot y\cdot z) - ((x-dx)\cdot y\cdot z)}{2dx} \\ = \lim_{dx\to 0} \frac{(x+dx)-(x-dx)}{2dx} yz &= \lim_{dx\to 0} \frac{2dx}{2dx} yz = yz\end{aligned}$$

We can apply the same method for each variable, the result will be the elements of the *gradient* ∇f

$$\frac{\partial f}{\partial x} = yz \qquad \frac{\partial f}{\partial y} = xz \qquad \frac{\partial f}{\partial z} = xy \tag{C.12}$$

The important thing to understand that in a computational graph, a multiplicative node, which takes N arbitrary parameters (or arguments), will have a *partial derivative* for each variable its output is depending on. In general if these derivatives are represented as a vector, then it is called the gradient ∇f of f. Also the value of the derivative will be the product of all variables except the one of which we are computing the influence of on the output.

Addition rule. Consider a value $x \cdot y + x \cdot z$ represented by $g(x, y, z)$. The change of g with respect to x is defined with the following shortened equation:

$$\frac{\partial g}{\partial x} = \lim_{dx \to 0} \frac{((x + dx)y + (x + dx)z) - ((x - dx)y + (x - dx)z)}{dx} = y + z \tag{C.13}$$

Notice that – in the terms of computational graphs – if a node contributes to other different operations (namely $x \cdot y$ and $x \cdot z$), than the derivative of each occurrence in *later* values will be summed up.

Chain rule. Let $f(x) = 2x+3$ and $g(f) = 5f$. Suppose that we would like to know the derivative of g with respect to x. At first we cannot do so, but there are two options: in the hope that substituting the value represented by f into g would not make the equation too complex we can unroll the references and rewrite $g(x) = 5 \cdot (2x + 3)$, or we could use the chain rule:

$$\frac{\partial g}{\partial x} = \lim_{dx \to 0} \frac{g(f(x + dx)) - g(f(x - dx))}{2dx}$$

Assume that $f(x + dx) - f(x - dx) \neq 0$.

$$\lim_{dx \to 0} \frac{g(f(x + dx)) - g(f(x - dx))}{f(x + dx) - f(x - dx)} \cdot \frac{f(x + dx) - f(x - dx)}{2dx}$$

$$\frac{\partial g}{\partial x} = \frac{\partial g}{\partial f} \cdot \frac{\partial f}{\partial x} \tag{C.14}$$

Which is a formula of the products of partial derivative of g, that treats f like a variable, and f explicitly operating on variable x. The derivative of g with respect to variable x is the product of the *local derivative* of g is $\frac{\partial g}{\partial f} = 5$ and $\frac{\partial f}{\partial x} = 2$ which equals $\frac{\partial g(x)}{\partial x} = \frac{\partial(5 \cdot (2x+3))}{\partial x} = 10$, the function strictly depending on x. The important message is that we can interchangeably use function values and variables with a constraint that at a point, in an arbitrary depth there must be a real variable. *Note:* the statement above stands for computations with *Acyclic Graph*, meaning that there should not be any feedbacks or loops – no definitions like $f(g), g(f)$ or $f(f)$. We will see that these rules play a very fundamental role in training networks.

Vector Notation. Before drilling deep into mathematical equations, a small reminder: the following vector and matrix formulation is just a special annotation, using the rules above, which helps to make clear both the definition, and the computations done by the network when it is implemented. The vectors with partially derivatives inside are just representing *real values*, arranged in a fancy way. Every vector and matrix defined in forward propagation, has its corresponding derivative w.r.t. the Loss. More trivially, if any value is depending on a list of variables $f(x_1, x_2 \cdots x_n) = f(\mathbf{x})$ (a vector) then there is a list of *partial derivatives* w.r.t. to f – forming the gradient $\nabla f = \left(\frac{\partial f}{x_1}, \frac{\partial f}{x_1} \cdots \frac{\partial f}{x_N}\right)$. *Notation*: When the vector notation is emphasized the variable name is conventionally written in bold font $\mathbf{x}$, or is underlined $\underline{x}$. Because later the indexing would become too crowded, we only use the indexed notation when it is necessary, otherwise using x.

The next step is formulating the dependency of multiple functions on multiple variables. As seen above, a multivariate function has its gradient vector – in the same fashion as the list of variables were organized into vectors $\mathbf{x}$, values composed of them can also form a vector $F = (f_1, f_2 \cdots f_M)$, composing a multivalued function depending on the same variables. Doing so yields a first-order derivative matrix, composed of gradient vectors $J = (\nabla f_1, \nabla f_2 \cdots \nabla f_M)^T$, called the *Jacobian of F*. The Jacobian has as many rows as output values F has, and the same number of columns of the variables that f_i is a function of.

$$J(F) = \begin{pmatrix} \nabla f_1 \\ \vdots \\ \nabla f_M \end{pmatrix} = \begin{pmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_N} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_M}{\partial x_1} & \cdots & \frac{\partial f_M}{\partial x_N} \end{pmatrix} \tag{C.15}$$

The matrix and vector operations (such as addition and inner product) that can be performed on the derivative *arrays* are identical defined in the inference section. That is important because product of derivatives introduced by the chain rule, can be applied as well for multidimensional array of derivatives too.

Fully Connected Layer

Return to the toy example and begin with the last layer, with 3 nodes. If a sample x is inferred $\mathcal{F}(x) = y$, then the response of the network will be a vector of $dim(y) = 3$. If we took the L_2 *distance* between the response and the goal y^*, then it would tell how far we are from the ideal, by a single scalar value. Since we want to minimize it, we have to adjust the parameters of the network, namely descend on the gradient slope. To get the small extent of the update we have to evaluate $\frac{\partial \mathcal{L}}{\partial \phi}$. Intuitively in the case we would like to correct the weights of a decision, it would require two things:

The original situation (the input of the l^{th} layer x_l), which the decision was made in.

The error on the decision – the derivative δ^l of the Loss with regards to the decision.

Since x_l is obtained via inference, what we have to calculate is δ^l for the l^{th} layer in order to acquire the parameter gradient. The first step is to evaluate the δ_L, or the *error* of the last layer's response y^3, namely $\delta^3 = \nabla_{y^3}\mathcal{L}$. Begin with the first element:

$$\delta_1^3 = \frac{\partial \mathcal{L}}{\partial y_1} = \frac{\partial}{\partial y_1}\frac{1}{2}\left((y_1^* - y_1)^2 + (y_2^* - y_2)^2 + (y_3^* - y_3)^2\right) = y_1 - y_1^*$$

Expanding it to the whole array:

$$\delta^3 = \begin{pmatrix} \frac{\partial \mathcal{L}}{\partial y_1} \\ \\ \frac{\partial \mathcal{L}}{\partial y_2} \\ \\ \frac{\partial \mathcal{L}}{\partial y_3} \end{pmatrix} = \begin{pmatrix} \frac{\partial}{\partial y_1} \frac{1}{2} \sum_i (y_i^* - y_i)^2 \\ \\ \frac{\partial}{\partial y_2} \frac{1}{2} \sum_i (y_i^* - y_i)^2 \\ \\ \frac{\partial}{\partial y_3^3} \frac{1}{2} \sum_i (y_i^* - y_i)^2 \end{pmatrix} = \begin{pmatrix} y_1 - y_1^* \\ \\ y_2 - y_2^* \\ \\ y_3 - y_3^* \end{pmatrix}$$

Consider the following notation: $\delta_i^3 = (y_i^* - y_i)$, called parametric vector notation. Writing arrays in this way, saves a lot of space. However, when this notation gets jammed with indexes, it is useful to write down explicitly the whole array for clarification.

The last layer Now we have exact values of δ^3 and x^3, so we can calculate how should the weights in W^3 be changed in order to get a better network. Applying the differentiation rules for each weight (forming a matrix) of the layer will result in a derivative for each weight (also forming a matrix). Taking the first perceptron of the layer, it has a weight $W_1^3 = (W_{1,1}, W_{1,2}, W_{1,3}, W_{1,4})$ for each output of the previous layer.

Suppose that this neuron had to tell how rounded is the object on an image sample, and the i^{th} neuron of the 2^{nd} layer fires when it recognizes sharp edges. Of course it would be bad if our neuron had a large weight on x_i^3. If this node performs poorly because of x_i^3, then it would contribute a lot to the Loss function while inferring sharp objects, with its output y_1 being far away from y_1^*, resulting in a positive δ_1^3. In case of $x_i^3 = 0$, the weight $W_{1,i}$ has nothing to do with the error δ_1^3 of the neuron. Notice that the error of each weight (w.r.t $\mathcal{L}$) should be proportional to the error and the input as well: $\frac{\partial \mathcal{L}}{\partial w_{1,i}} = x_i^3 \cdot \delta_1^3$. However it can be also derived in terms of the differentiation rules:

$$\frac{\partial \mathcal{L}}{\partial W_{1,i}} = \frac{\partial \mathcal{L}}{\partial y_1^3} \cdot \frac{\partial y_1^3}{\partial W_{1,i}} = \delta_1^3 \cdot x_i^3$$

Considering the parameter update. assume that a picture of an origami sculpture (a very edgy one) was inferred and the i^{th} node of the second layer worked correctly. Both x_i^3 and δ_1^3 is positive, however the weight should be decreased: that is why we will take the negative of the derivative for updating $w_{1,i}$. Substituting $i = 1, 2, 3, 4$ into $\frac{\partial \mathcal{L}}{\partial w_{1,i}}$ yields a gradient of $\mathcal{L}$ with regards to the weights of the first neuron of the last layer. Doing so for each neurons in the layer would result in 3 gradient vectors $\nabla_{W_1}\mathcal{L}$, $\nabla_{W_2}\mathcal{L}$ and $\nabla_{W_3}\mathcal{L}$ which is practically stacked to make a matrix which can be later added element-wisely to the weight matrix W.

$$\nabla_W \mathcal{L} = \begin{pmatrix} \frac{\partial \mathcal{L}}{\partial W_{1,1}} & \frac{\partial \mathcal{L}}{\partial W_{1,2}} & \frac{\partial \mathcal{L}}{\partial W_{1,3}} & \frac{\partial \mathcal{L}}{\partial W_{1,4}} \\ \\ \frac{\partial \mathcal{L}}{\partial W_{2,1}} & \frac{\partial \mathcal{L}}{\partial W_{2,2}} & \frac{\partial \mathcal{L}}{\partial W_{2,3}} & \frac{\partial \mathcal{L}}{\partial W_{2,4}} \\ \\ \frac{\partial \mathcal{L}}{\partial W_{3,1}} & \frac{\partial \mathcal{L}}{\partial W_{3,2}} & \frac{\partial \mathcal{L}}{\partial W_{3,3}} & \frac{\partial \mathcal{L}}{\partial W_{3,4}} \end{pmatrix} = \begin{pmatrix} x_1\delta_1 & x_2\delta_1 & x_3\delta_1 & x_4\delta_1 \\ \\ x_1\delta_2 & x_2\delta_2 & x_3\delta_2 & x_4\delta_2 \\ \\ x_1\delta_3 & x_2\delta_3 & x_3\delta_3 & x_4\delta_3 \end{pmatrix}$$

The last part of the equation can be also expressed as:

$$\nabla_W \mathcal{L} = \left(\frac{\partial \mathcal{L}}{\partial W_{i,j}} \right) = (x_j \delta_i) = x \wedge \delta \tag{C.16}$$

Where $\wedge$ denotes the *outer product* operator. The gradient of the bias is simply $\nabla_{b^L} \mathcal{L} = \delta^L$.

The $(L-1)^{th}$ layer. Gradient descent can be applied on networks with more than one layer, however continuing the example of the image descriptor network requires a bit more abstraction. In the previous explanation we assumed that the i^{th} neuron of the second layer worked properly. In general cases this assumption is incorrect, if the whole network is initialized at once. If we think about that also the mentioned neuron in the *last hidden layer* should be trained, then we can apply the same method with a 4 dimensional $\delta^{(L-1)}$ and with a 5 dimensional $x^{(L-1)}$ input.

Note: Capital L is representing the number of layers. In the toy example $L = 3$.
Acquiring the $\delta^{(L-1)}$ is where the chain rule (C.14) steps into the scene. While in the example before we could find an intuitive workaround, in this case it would be quite strained, since $\mathcal{L}$ does not depend directly on $y_{(L-1)}$. Utilizing the chain rule:

$$\delta^{(L-1)} = \frac{\partial \mathcal{L}}{\partial y^{(L-1)}} = \frac{\partial \mathcal{L}}{\partial y^L} \cdot \frac{\partial y^L}{\partial y^{(L-1)}} = \delta^L \cdot J(F_L) \tag{C.17}$$

Where $J(F_L)$ denotes the *Jacobian* (C.15) of the function representing the projection of the last layer $F_L : \mathbb{R}^4 \mapsto \mathbb{R}^3$. It is a map of how each input variable affects the output of the layer. In general: the Jacobian numerically can be represented as a 3×4 matrix, analytically as a *local derivative* of function F. In case of *Fully Connected* layers the Jacobian is simply the weight matrix $F(F_l) = W_l$ (the bias drops out here).

For derivative of scalar values ($\mathbb{R}$) the *product* operator is well defined, and it can be expanded the same way to derivatives of multidimensional values as regular *inner product* of the values they are composed of. *Note*: In order to stay consistent with the dimensions of the computation, we have to switch sides of the matrix multiplication defined in (C.2):

$$\begin{aligned} \delta_j^{(L-1)} &= \sum_j \delta_i^L \, W_{i,j}^L \\ \delta^{(L-1)} &= \delta^L \cdot W^L \\ [4] &= [3] \cdot [3 \times 4] \end{aligned} \tag{C.18}$$

Backpropagation

The backpropagation algorithm, first described by Werbos *et. al* [80]

The l^{th} layer. The contribution to the Loss of the general l^{th} layer δ^l can be retrieved by unfolding $\frac{\partial \mathcal{L}}{\partial y^l}$ applying the chain rule, namely the backpropagation:

$$\begin{aligned}
\delta^l &= \frac{\partial \mathcal{L}}{\partial y^l} = \frac{\partial \mathcal{L}}{\partial y^L} \cdot \frac{\partial y^L}{\partial y^{(L-1)}} \quad \cdots \quad \frac{\partial y^{(l+1)}}{\partial y^l} \\
\delta^l &= \delta^L \cdot J(F_L) \cdot J(F_{(L-1)}) \quad \cdots \quad J(F_{(l-1)}) \\
[dim(l)] &= [dim(L)] \cdot [dim(L) \times dim(L-1)] \cdots [dim(l+1) \times dim(l)]
\end{aligned} \tag{C.19}$$

Note: the order of evaluating these derivatives is theoretically irrelevant, however computationally there is an opportunity to implement it in two ways [30]:

forward-mode differentiation: evaluating in the order of layers is efficient in cases where the output of the network is much larger than the input

reverse-mode differentiation: evaluating in reversed order for networks with fewer outputs than inputs.

As pointed out in the thesis, the former would require $(L-l)$ times evaluating a *matrix-matrix* product and one *vector-matrix* operation, while the latter would require $(L-l)$ times evaluating a *vector-matrix* product and one *matrix-matrix* at the end. Using forward-mode differentiation does not require keeping transient activations y^l, however it is computationally costly. Using backward-mode differentiation does not strain the CPU, but the memory. It is because if $J(F_l)$ is not linear, then it requires the input x_l to evaluate the first order derivatives.

Activation Layer

Though the activation layer operates on the input, it has no adjustable parameter. Since we have to evaluate δ for layers behind activation layers, the error must pass through $J(F_{activation}$ as well. Luckily activation layers applies a scalar function element-wise on the inputs, the Jacobian of the composite function F has a special attribute: it is **diagonal**, meaning that:

$$J(F) = \frac{\partial F_i}{\partial x_j} = \begin{cases} \frac{\partial f(x_i)}{\partial x_i} & i = j \\ 0 & i \neq j \end{cases}$$

We can exploit this function when implementing activation layers, by simply using element-wise product $\frac{\partial f(x_i)}{\partial x_i}$ instead of a matrix multiplication. The mentioned activation functions (C.3, C.4, C.5, C.6), are differentiable *almost everywhere*, the corresponding derivatives:

$$(\text{ReLU})' := \text{Heaviside}(x) \tag{C.20}$$

$$(\text{TanH})' := 1 - \tanh(x)^2 \tag{C.21}$$

$$(\text{SP})' := \frac{1}{1+e^{-x}} \tag{C.22}$$

$$(\text{Log})' := \text{Logistic}(x) \cdot (1 - \text{Logistic}(x)) \tag{C.23}$$

Pooling layers

Pooling layers are decreasing the parameter space with a fixed down-sampling projection which operates independently on tiles of input. In general pooling layers are moving a `N`-sized window, or a tile through the data taking `stride` steps. Data in each window is collapsed by an aggregation function, such as `mean()` or `max()` lowering the computational complexity of the next layers processes. Implementation of this layer can be a bottleneck of inference, since the operations carried out in each window may rely on the same input values if `N` > `stride` i.e. windows overlap. In my library a single `max-pool` layer is implemented with fixed parameters: `stride` = `N` = 2. Further information see [32] `~/convolutional-networks/#pool`.

k-WTA

k-Winner Takes All layers are similar to pooling layers, without preserving spatial information. They function as a threshold of the previous layer, only passing those activations which have the largest magnitude among the others. Gradient is also distributed between those neurons which were passed by the *k-WTA* layer. Mainly used in fully-connected networks for reducing computational complexity. This layer was applied when generating MNIST visualization. For further information see [45].

Dropout

Dropout layers target the problem of overfitting, being considered as a method of regularization. In theory they reduce complex co-adaptation via model averaging, by generating different networks (using the same parameters) in each forward pass [23]. Practically they drop activations of the previous layer (with random *Bernoulli(p)* probability) when propagating information, forcing succeeding layers to not depend on an individual neuron of the layer before. Computationally it is very efficient to create different, so called **sub-networks**: both for forward and backward pass a single *Boolean array* mask is used. *Note*: Dropout layers are reducing the overall magnitude of the activation of the previous layer, considering that latent layers would be *saturated* when dropout layers were removed after training, the activations passed by this layer are multiplied by the constant $\frac{1}{p}$.

DropConnect. A similar concept, introduced by Li Wan *et al.* [78]: instead of dropping the activation of the previous layer, the weights of the layer are canceled out in the same fashion. In my implementation they can be only applied to fully-connected layers, where a *Boolean matrix* mask is used.

Output

On the top of every network there should be a dedicated layer, which arranges the networks response. If a *(sample, label)* pair was inferred it also evaluates the result and output a *(response, loss)* pair. Therefore the output layer will determine how the network will be trained, and the response interpreted

Softmax. In case of classification, when the network has to decide which set was the input sampled from, then we can transform the response into a categorical probability distribution, i.e. how confident each neuron in the last layer in classifying the input as their 'own'. For that, we use **one-hot** notation, where $y_i^* = [0, 0 \cdots 1 \cdots 0]$ (with only one non-zero element) represents the *ground-truth* label, the i^{th} class. The response will be turned into a discrete density function, e. g. $= [0.1,\ 0.6,\ 0.2,\ 0.1]$ can be interpreted as the following: the input x is categorized as a sample from the second class with 60% confidence. Notice that $|p|_1 = \sum_i y_i = 1$ constraint should be satisfied. To transform a real valued N dimensional vector $y \in \mathbb{R}^N$ to the mentioned probability distribution $p \in \Omega^N$ we apply the *Softmax* function:

$$p_i = \frac{\exp(y_i)}{|\exp(y)|_1} \tag{C.24}$$

Conventionally the corresponding Loss function is the **log-loss** or **cross-entropy** function:

$$\mathcal{L}(y) = -\sum_i y_i^* \log p_i \tag{C.25}$$

and the derivation of its gradient w.r.t. the original response of the last layer y, is the following (based on [28]):

$$\begin{aligned}
\frac{\partial \mathcal{L}}{\partial y_i} &= -\sum_k y_j^* \frac{\partial \log p_j}{\partial y_i} = -\sum_k \frac{y_j^*}{p_j} \frac{\partial p_j}{\partial y_i} \\
&= -y_i^* (1 - p_i) - \sum_{j \neq i} \frac{y_j^*}{p_j} (p_i p_j) \\
&= -y_i^* - y_i^* p_i + \sum_{j \neq i} y_j^* p_i \\
&= -y_i^* + \sum_j y_j^* p_i \\
&= p_i \left(\sum_j y_j^* \right) - y_i^* \\
&= p_i - y_i^*
\end{aligned} \tag{C.26}$$

The trick of the derivation is that y^* has only one non-zero element which is 1, and the lemma:

$$\frac{\partial p_j}{\partial y_i} = \begin{cases} p_i(1 - p_i) & i = j \\ -p_i p_j & i \neq j \end{cases}$$

Bibliography

[1] *A Support Vector Machine approach for reliable detection of atrial fibrillation events - IEEE Conference Publication.* URL: `http://ieeexplore.ieee.org/abstract/document/6713560/` (visited on 11/18/2017).

[2] *AF Classification from a short single lead ECG recording: the PhysioNet/Computing in Cardiology Challenge 2017.* URL: `https://physionet.org/challenge/2017/` (visited on 05/08/2017).

[3] R Alcaraz et al. "Wavelet sample entropy: A new approach to predict termination of atrial fibrillation". In: *Computers in Cardiology, 2006.* IEEE. 2006, pp. 597–600.

[4] *Atrial fibrillation detection by heart rate variability in Poincare plot | BioMedical Engineering OnLine | Full Text.* URL: `https://biomedical-engineering-online.biomedcentral.com/articles/10.1186/1475-925X-8-38` (visited on 11/18/2017).

[5] *Automatic online detection of atrial fibrillation based on symbolic dynamics and Shannon entropy | BioMedical Engineering OnLine | Full Text.* URL: `https://biomedical-engineering-online.biomedcentral.com/articles/10.1186/1475-925X-13-18` (visited on 11/18/2017).

[6] Saeed Babaeizadeh et al. "Improvements in atrial fibrillation detection for real-time monitoring". eng. In: *Journal of Electrocardiology* 42.6 (Dec. 2009), pp. 522–526. ISSN: 1532-8430. DOI: `10.1016/j.jelectrocard.2009.06.006`.

[7] Evgeny Burnaev, Pavel Erofeev, and Artem Papanov. "Influence of Resampling on Accuracy of Imbalanced Classification". In: *arXiv:1707.03905 [cs, stat]* (Dec. 2015). arXiv: 1707.03905, p. 987521. DOI: `10.1117/12.2228523`. URL: `http://arxiv.org/abs/1707.03905` (visited on 11/16/2017).

[8] Emre Çakır et al. "Convolutional Recurrent Neural Networks for Polyphonic Sound Event Detection". In: *IEEE/ACM Transactions on Audio, Speech, and Language Processing* 25.6 (June 2017). arXiv: 1702.06286, pp. 1291–1303. ISSN: 2329-9290, 2329-9304. DOI: `10.1109/TASLP.2017.2690575`. URL: `http://arxiv.org/abs/1702.06286` (visited on 11/16/2017).

[9] Marta Carrara et al. "Heart rate dynamics distinguish among atrial fibrillation, normal sinus rhythm and sinus rhythm with frequent ectopy". eng. In: *Physiological Measurement* 36.9 (Sept. 2015), pp. 1873–1888. ISSN: 1361-6579. DOI: `10.1088/0967-3334/36/9/1873`.

[10] Junyoung Chung et al. "Empirical evaluation of gated recurrent neural networks on sequence modeling". In: *arXiv preprint arXiv:1412.3555* (2014).

[11] ROBERTA COLLOCA. *Implementation and testing of atrial fibrillation detectors for a mobile phone application.* eng. Laurea Magistrale / Specialistica. Apr. 2013. URL: `https://www.politesi.polimi.it/handle/10589/78201` (visited on 11/18/2017).

[12] Deeptankar DeMazumder et al. "Dynamic analysis of cardiac rhythms for discriminating atrial fibrillation from lethal ventricular arrhythmias". eng. In: *Circulation. Arrhythmia and Electrophysiology* 6.3 (June 2013), pp. 555–561. ISSN: 1941-3084. DOI: `10.1161/CIRCEP.113.000034`.

[13] Jia Deng et al. "Imagenet: A large-scale hierarchical image database". In: *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on.* IEEE. 2009, pp. 248–255.

[14] Xiaochuan Du et al. "A Novel Method for Real-Time Atrial Fibrillation Detection in Electrocardiograms Using Multiple Parameters". In: *Annals of Noninvasive Electrocardiology* 19.3 (2014), pp. 217–225.

[15] John Duchi, Elad Hazan, and Yoram Singer. "Adaptive subgradient methods for online learning and stochastic optimization". In: *Journal of Machine Learning Research* 12.Jul (2011), pp. 2121–2159.

[16] Yaroslav Ganin et al. "Domain-Adversarial Training of Neural Networks". In: *arXiv:1505.07818 [cs, stat]* (May 2015). arXiv: 1505.07818. URL: `http://arxiv.org/abs/1505.07818` (visited on 05/07/2017).

[17] Manuel García et al. "Application of the relative wavelet energy to heart rate independent detection of atrial fibrillation". In: *Computer methods and programs in biomedicine* 131 (2016), pp. 157–168.

[18] Leon A Gatys et al. "Controlling perceptual factors in neural style transfer". In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR).* 2017.

[19] Ross Girshick et al. "Rich feature hierarchies for accurate object detection and semantic segmentation". In: *Proceedings of the IEEE conference on computer vision and pattern recognition.* 2014, pp. 580–587.

[20] *Global, regional, and national life expectancy, all-cause mortality, and cause-specific mortality for 249 causes of death, 1980-2015: a systematic ... - PubMed - NCBI.* URL: `https://www.ncbi.nlm.nih.gov/pubmed/27733281` (visited on 11/20/2017).

[21] Kaiming He et al. "Deep residual learning for image recognition". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition.* 2016, pp. 770–778.

[22] Kaiming He et al. “Identity Mappings in Deep Residual Networks”. In: *arXiv:1603.05027 [cs]* (Mar. 2016). arXiv: 1603.05027. URL: `http://arxiv.org/abs/1603.05027` (visited on 05/22/2017).

[23] Geoffrey E Hinton et al. “Improving neural networks by preventing co-adaptation of feature detectors”. In: *arXiv preprint arXiv:1207.0580* (2012).

[24] Sepp Hochreiter and Jürgen Schmidhuber. “Long short-term memory”. In: *Neural computation* 9.8 (1997), pp. 1735–1780.

[25] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. “Multilayer feedforward networks are universal approximators”. In: *Neural networks* 2.5 (1989), pp. 359–366.

[26] Chao Huang et al. “A novel method for detection of the transition between atrial fibrillation and sinus rhythm”. eng. In: *IEEE transactions on bio-medical engineering* 58.4 (Apr. 2011), pp. 1113–1119. ISSN: 1558-2531. DOI: `10.1109/TBME.2010.2096506`.

[27] Gao Huang et al. “Densely Connected Convolutional Networks”. In: *arXiv:1608.06993 [cs]* (Aug. 2016). arXiv: 1608.06993. URL: `http://arxiv.org/abs/1608.06993` (visited on 11/20/2017).

[28] Moos Hueting. Accessed: June 5, 2025. URL: `http://math.stackexchange.com/questions/945871`.

[29] Sergey Ioffe and Christian Szegedy. “Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift”. In: *arXiv:1502.03167 [cs]* (Feb. 2015). arXiv: 1502.03167. URL: `http://arxiv.org/abs/1502.03167` (visited on 05/09/2017).

[30] Andrej Karpathy. “Connecting Images and Natural Language”. PhD thesis. Stanford University, 2016.

[31] Andrej Karpathy. “Andrej Karpathy blog Hacker’s guide to Neural Networks”. Accessed: June 5, 2025. URL: `http://karpathy.github.io/neuralnets/`.

[32] Andrej Karpathy, Fei-Fei Li, and Justin Johnson. “CS231n Convolutional Neural Networks for Visual Recognition.” Accessed: June 5, 2025. 2016. URL: `http://cs231n.github.io/`.

[33] Diederik Kingma and Jimmy Ba. “Adam: A method for stochastic optimization”. In: *arXiv preprint arXiv:1412.6980* (2014). URL: `https://arxiv.org/abs/1412.6980` (visited on 05/08/2017).

[34] Diederik Kingma and Jimmy Ba. “Adam: A method for stochastic optimization”. In: *arXiv preprint arXiv:1412.6980* (2014).

[35] Günter Klambauer et al. “Self-Normalizing Neural Networks”. In: *arXiv:1706.02515 [cs, stat]* (June 2017). arXiv: 1706.02515. URL: `http://arxiv.org/abs/1706.02515` (visited on 11/20/2017).

[36] Alex Krizhevsky and Geoffrey Hinton. “Learning multiple layers of features from tiny images”. In: (2009).

[37] Siddharth Krishna Kumar. “On weight initialization in deep neural networks”. In: *arXiv:1704.08863 [cs]* (Apr. 2017). arXiv: 1704.08863. URL: `http://arxiv.org/abs/1704.08863` (visited on 11/20/2017).

[38] Steven Ladavich and Behnaz Ghoraani. “Rate-independent detection of atrial fibrillation by statistical modeling of atrial activity”. In: *Biomedical Signal Processing and Control* 18 (2015), pp. 274–281.

[39] Douglas E. Lake and J. Randall Moorman. “Accurate estimation of entropy in very short physiological time series: the problem of atrial fibrillation detection in implanted ventricular devices”. eng. In: *American Journal of Physiology. Heart and Circulatory Physiology* 300.1 (Jan. 2011), H319–325. ISSN: 1522-1539. DOI: `10.1152/ajpheart.00561.2010`.

[40] Martin Längkvist, Lars Karlsson, and Amy Loutfi. “A review of unsupervised feature learning and deep learning for time-series modeling”. In: *Pattern Recognition Letters* 42 (2014), pp. 11–24.

[41] Yann LeCun, Corinna Cortes, and Christopher JC Burges. *The MNIST database of handwritten digits*. 1998.

[42] Yann LeCun, Yoshua Bengio, et al. “Convolutional networks for images, speech, and time series”. In: *The handbook of brain theory and neural networks* 3361.10 (1995), p. 1995.

[43] Qiao Li et al. “Signal processing and feature selection preprocessing for classification in noisy healthcare data”. In: Aug. 2016. ISBN: 9781849199780.

[44] David T. Linker. “Accurate, Automated Detection of Atrial Fibrillation in Ambulatory Recordings”. eng. In: *Cardiovascular Engineering and Technology* 7.2 (June 2016), pp. 182–189. ISSN: 1869-4098. DOI: `10.1007/s13239-016-0256-z`.

[45] Qingshan Liu and Jun Wang. “Two k-winners-take-all networks with discontinuous activation functions”. In: *Neural Networks* 21.2 (2008), pp. 406–413.

[46] Jonathan Long, Evan Shelhamer, and Trevor Darrell. “Fully Convolutional Networks for Semantic Segmentation”. In: *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2015.

[47] Luca Mainardi, Leif Sörnmo, and Sergio Cerutti. “Understanding Atrial Fibrillation: The Signal Processing Contribution, Part I”. In: *Synthesis Lectures on Biomedical Engineering* 3.1 (Jan. 2008), pp. 1–129. ISSN: 1930-0328. DOI: `10.2200/S00152ED1V01Y200809BME02` URL: `http://www.morganclaypool.com/doi/abs/10.2200/S00152ED1V01Y200809BME024` (visited on 11/18/2017).

[48] Luca Mainardi, Leif Sörnmo, and Sergio Cerutti. “Understanding Atrial Fibrillation: The Signal Processing Contribution, Part II”. In: *Synthesis Lectures on Biomedical Engineering* 3.1 (Jan. 2008), pp. 1–139. ISSN: 1930-0328. DOI: `10.2200/S00153ED1V01Y200809BME025`. URL: `http://www.morganclaypool.com/doi/abs/10.2200/S00153ED1V01Y200809BME025` (visited on 11/18/2017).

[49] Pankaj Malhotra et al. "Long short term memory networks for anomaly detection in time series". In: *Proceedings*. Presses universitaires de Louvain. 2015, p. 89.

[50] Tomas Mikolov et al. "Recurrent neural network based language model." In: *Interspeech*. Vol. 2. 2010, p. 3. URL: `http://www.fit.vutbr.cz/research/groups/speech/servite/2010/rnnlm_mikolov.pdf` (visited on 05/09/2017).

[51] Roni Mittelman. "Time-series modeling with undecimated fully convolutional neural networks". In: *arXiv preprint arXiv:1508.00317* (2015).

[52] Yasumasa Miyamoto and Kyunghyun Cho. "Gated Word-Character Recurrent Language Model". In: *arXiv:1606.01700 [cs]* (June 2016). arXiv: 1606.01700. URL: `http://arxiv.org/abs/1606.01700` (visited on 05/07/2017).

[53] Michael Nielsen. "Neural Networks and Deep Learning". Accessed: June 5, 2025. 2016. URL: `http://neuralnetworksanddeeplearning.com/`.

[54] Chris Olah. *Understanding LSTM Networks – colah's blog*. URL: `http://colah.github.io/posts/2015-08-Understanding-LSTMs/` (visited on 05/07/2017).

[55] Aaron van den Oord et al. "WaveNet: A Generative Model for Raw Audio". In: *arXiv:1609.03499 [cs]* (Sept. 2016). arXiv: 1609.03499. URL: `http://arxiv.org/abs/1609.03499` (visited on 11/19/2017).

[56] *Optimal parameters study for sample entropy-based atrial fibrillation organization analysis - Surrey Research Insight Open Access*. URL: `http://epubs.surrey.ac.uk/124492/` (visited on 11/18/2017).

[57] Julien Oster and Gari D. Clifford. "Impact of the presence of noise on RR interval-based atrial fibrillation detection". eng. In: *Journal of Electrocardiology* 48.6 (Dec. 2015), pp. 947–951. ISSN: 1532-8430. DOI: `10.1016/j.jelectrocard.2015.08.013`.

[58] Andrius Petrėnas, Vaidotas Marozas, and Leif Sörnmo. "Low-complexity detection of atrial fibrillation in continuous long-term monitoring". eng. In: *Computers in Biology and Medicine* 65 (Oct. 2015), pp. 184–191. ISSN: 1879-0534. DOI: `10.1016/j.compbiomed.2015.01.019`.

[59] Andrius Petrenas et al. "An echo state neural network for QRST cancellation during atrial fibrillation". In: *IEEE Transactions on Biomedical Engineering* 59.10 (2012), pp. 2950–2957.

[60] Andrius Petrėnas et al. "Detection of occult paroxysmal atrial fibrillation". eng. In: *Medical & Biological Engineering & Computing* 53.4 (Apr. 2015), pp. 287–297. ISSN: 1741-0444. DOI: `10.1007/s11517-014-1234-y`.

[61] Helmut Pürerfellner et al. "P-wave evidence as a method for improving algorithm to detect atrial fibrillation in insertable cardiac monitors". In: *Heart Rhythm* 11.9 (2014), pp. 1575–1583.

[62] Pranav Rajpurkar et al. "Cardiologist-Level Arrhythmia Detection with Convolutional Neural Networks". In: *arXiv:1707.01836 [cs]* (July 2017). arXiv: 1707.01836. URL: `http://arxiv.org/abs/1707.01836` (visited on 11/16/2017).

[63] James A Reiffel et al. “Practice patterns among United States cardiologists for managing adults with atrial fibrillation (from the AFFECTS Registry)”. In: *The American journal of cardiology* 105.8 (2010), pp. 1122–1129.

[64] Juan Ródenas et al. “Wavelet Entropy Automatically Detects Episodes of Atrial Fibrillation from Single-Lead Electrocardiograms”. In: *Entropy* 17.9 (2015), pp. 6179–6199.

[65] Gavin A Rummery and Mahesan Niranjan. *On-line Q-learning using connectionist systems*. University of Cambridge, Department of Engineering, 1994.

[66] David Saad. *On-line learning in neural networks*. Vol. 17. Cambridge University Press, 2009.

[67] Adam Santoro et al. “One-shot Learning with Memory-Augmented Neural Networks”. In: *arXiv:1605.06065 [cs]* (May 2016). arXiv: 1605.06065. URL: `http://arxiv.org/abs/1605.06065` (visited on 05/07/2017).

[68] Shantanu Sarkar, David Ritscher, and Rahul Mehra. “A detector for a chronic implantable atrial tachyarrhythmia monitor”. eng. In: *IEEE transactions on biomedical engineering* 55.3 (Mar. 2008), pp. 1219–1224. ISSN: 0018-9294. DOI: `10.1109/TBME.2007.903707`.

[69] Jonathan Richard Shewchuk. *An introduction to the conjugate gradient method without the agonizing pain*. 1994.

[70] Karen Simonyan and Andrew Zisserman. “Very Deep Convolutional Networks for Large-Scale Image Recognition”. In: *arXiv:1409.1556 [cs]* (Sept. 2014). arXiv: 1409.1556. URL: `http://arxiv.org/abs/1409.1556` (visited on 05/09/2017).

[71] Ilya Sutskever et al. “On the importance of initialization and momentum in deep learning.” In: *ICML (3)* 28 (2013), pp. 1139–1147.

[72] K. Tateno and L. Glass. “Automatic detection of atrial fibrillation using the coefficient of variation and density histograms of RR and deltaRR intervals”. eng. In: *Medical & Biological Engineering & Computing* 39.6 (Nov. 2001), pp. 664–671. ISSN: 0140-0118.

[73] Tomás Teijeiro et al. “Arrhythmia Classification from the Abductive Interpretation of Short Single-Lead ECG Records”. In: *arXiv:1711.03892 [cs]* (Nov. 2017). arXiv: 1711.03892. URL: `http://arxiv.org/abs/1711.03892` (visited on 11/20/2017).

[74] Tijmen Tieleman and Geoffrey Hinton. “Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude”. In: *COURSERA: Neural Networks for Machine Learning* 4.2 (2012).

[75] Andreas Veit, Michael Wilber, and Serge Belongie. “Residual Networks Behave Like Ensembles of Relatively Shallow Networks”. In: *arXiv:1605.06431 [cs]* (May 2016). arXiv: 1605.06431. URL: `http://arxiv.org/abs/1605.06431` (visited on 05/09/2017).

[76] Oriol Vinyals et al. “Matching Networks for One Shot Learning”. In: (June 2016). (Visited on 05/07/2017).

[77] Li Wan et al. “Regularization of neural networks using dropconnect”. In: *Proceedings of the 30th International Conference on Machine Learning (ICML-13).* 2013, pp. 1058–1066. URL: `http://machinelearning.wustl.edu/mlpapers/papers/icml2013_wan13` (visited on 05/08/2017).

[78] Li Wan et al. “Regularization of neural networks using dropconnect”. In: *Proceedings of the 30th International Conference on Machine Learning (ICML-13).* 2013, pp. 1058–1066.

[79] Zhiguang Wang, Weizhong Yan, and Tim Oates. “Time Series Classification from Scratch with Deep Neural Networks: A Strong Baseline”. In: *arXiv preprint arXiv:1611.06455* (2016).

[80] Paul John Werbos. *The roots of backpropagation: from ordered derivatives to neural networks and political forecasting.* Vol. 1. John Wiley & Sons, 1994.

[81] Jason Yosinski et al. “Understanding neural networks through deep visualization”. In: *arXiv preprint arXiv:1506.06579* (2015).

[82] Fisher Yu and Vladlen Koltun. “Multi-Scale Context Aggregation by Dilated Convolutions”. In: *arXiv:1511.07122 [cs]* (Nov. 2015). arXiv: 1511.07122. URL: `http://arxiv.org/abs/1511.07122` (visited on 11/19/2017).

[83] Morteza Zabihi et al. “Detection of Atrial Fibrillation in ECG Hand-held Devices using a Random Forest Classifier”. In: Sept. 2017.

[84] Matthew D Zeiler and Rob Fergus. “Visualizing and understanding convolutional networks”. In: *European Conference on Computer Vision.* Springer. 2014, pp. 818–833.