ECG Arrhythmia Detection and Classification - A Comparison of Recurrent and Convolutional Approaches

Tobias Weber
Ludwig-Maximilians-Universität München
Munich, Germany
t.weber@campus.lmu.de

***Abstract*—Cardiovascular diseases and arrhythmias are two of the most spread death causes worldwide. The automatic classification of ECGs, which helps with early diagnosis is an active area of research in machine learning. The proposed paper examines the question whether recurrent or convolutional neural networks are a better fit for this task. Multiple experiments with varying layers, batch sizes, units or regularization were conducted. According to the results a CNN in comparison to LSTMs puts less stress on the system, converges in a fraction of time and achieves equivalent test performance.**

I. Introduction

Cardiovascular diseases are responsible for 17.9 million fatalities yearly, which represent 31% of all deaths worldwide (cf. WHO). A respective number of patients would benefit from identifying diseases in early stages, which allows successful medical treatment. The 12-lead *electrocardiogram* (ECG), which monitors the electrical activity of the heart, depicts a crucial medium for diagnosing *arrhythmias* and other anomalies. Interpretation of critical diseases is still subject to human experts but modern machine learning approaches can help to recognize patterns and therefore are able to assist in decision making. The task of detecting anomalies via computerized ECGs is still an active area of research. Different architectures and approaches have been evaluated on various datasets, e.g. *Gradient Boosting* [1] or *Neural Networks* [2]. High capacity function approximators in the field of deep learning enable dedicated end-to-end architectures [3] or combination of recurrent and convolutional units [4]. ECGAN [5] on the other hand focuses on detecting anomalies by employing *Generative Adversarial Networks*, which mark out-of-distribution ECGs. In summary a variety of machine learning techniques is utilized in current literature to solve the domain of arrhythmia detection. The proposed paper examines the question whether to rely on recurrent or convolutional approaches for this particular task by conducting experiments with different configurations on a fixed dataset. Additionally the process of transforming raw data to clean input for training is depicted.

II. Theoretical Background

A. Long short-term memory

Long Short-term Memories (LSTM) [6] belong to the family of *Recurrent Neural Networks* (RNN), which are designed to handle temporal sequences of dynamic length. One of the basic mechanics of RNNs is the self-connection that propagates information over a range of timesteps. However, when training is conducted through *Backpropagation through time* (BPTT) the updates are prone to vanishing or exploding gradients due to repeated multiplication with small or large weights. LSTMs mitigate this issue by adding a *Cellstate* C, which serves for long-term dependencies, while the *hidden state* h is kept for short-term memorization.

A conceptualized view of the information flow within a LSTM unit is shown in Figure 1. Equation (1) depicts the concrete calculation of the target values. In this case $x_t \in \mathbb{R}^d$ is a single d-dimensional observation at timestep t. Further the hidden state $h \in \mathbb{R}^h$ marks the dimension of the spanned space of the LSTM unit and $\odot$ denotes the Hadamard-product. The cell itself consists of multiple gates that serve different purposes. The *forget gate* f_t combines an affine linear transformation of x_t and h_{t-1} with a sigmoidal function that reflects a mapping to (0,1) as non-linearity. As the name states, the resulting vector $\in \mathbb{R}^h$ indicates elements that will vanish from C_{t-1}. The *input gate* i_t performs the same kind of transformation as f_t with the different purpose of determining which values to add to C_{t-1}. $\tilde{C}_t$ serves as mapping to h-dimensional space with range (-1, 1). The *output gate* o_t again follows the same principles but with the intention of defining the properties which should be returned as h_t from the $tanh$-mapped C_t. The decisions made in the gate layers are eventually applied to the targets via element-wise multiplication. The cell propagates C_t and h_t to the next timestep and makes h_t available for further processing.

$$\begin{aligned} f_t &= \sigma(W_f x_t + U_f h_{t-1} + b_f) \\ i_t &= \sigma(W_i x_t + U_i h_{t-1} + b_i) \\ o_t &= \sigma(W_o x_t + U_o h_{t-1} + b_o) \\ \tilde{C}_t &= tanh(W_c x_t + U_c h_{t-1} + b_c) \\ C_t &= f_t \odot C_{t-1} + i_t \odot \tilde{C}_t \\ h_t &= o_t \odot tanh(C_t) \end{aligned} \tag{1}$$

In total the LSTM unit consists of various weight matrices and bias vectors, which results in a comparatively large number of trainable parameters. For $W \in \mathbb{R}^{h \times d}$, $U \in \mathbb{R}^{h \times h}$, $b \in \mathbb{R}^h$ and four internal layers the final number of parameters is

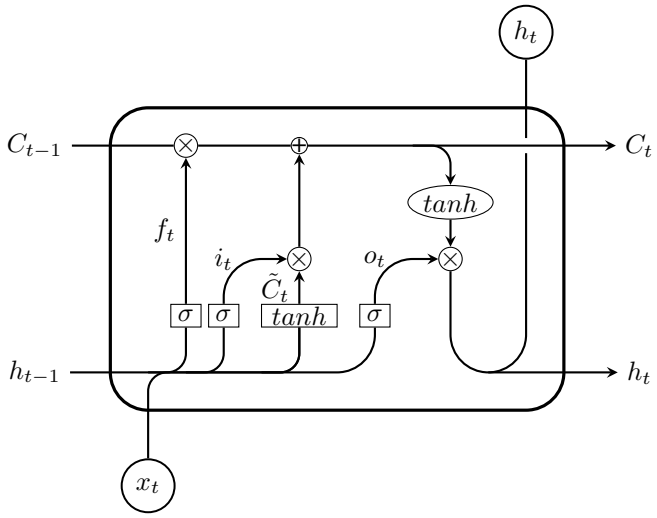


Fig. 1: Control flow of a LSTM unit (cf. [7])

$4(dh + h^2 + h)$, which scales quadratically to the internal space. A high number of parameters also indicates an extensive hypothesis space that is of advantage when approximating complicated functions but also comes with a tendency to overfit on small datasets and a rather high variance. In the last years LSTMs have succeeded in various time-series domains and text processing. An alternative way of tackling these topics, which is becoming increasingly popular, is by utilizing convolutional neural networks.

B. Convolutional Neural Networks

The general idea of *Convolutional Neural Networks* (CNN) is transforming the input into a different sophisticated (higher dimensional) representation by using filters. The technique is similar to applying *Sobel-Operators* with the difference, that the filters are not predefined but trained by backpropagation. This way the networks learns to extract patterns from the original source, which is helpful in classification tasks. Theoretically a fully-connected layer could be applied to solve the same task, but this would result in an infeasible number of weights even for small inputs. CNNs on the other hand use weight sharing due to the filter operations and are also translation-invariant due to local search. In comparison to RNNs the CNN cannot take inputs of variable length, which make them less flexible. Otherwise CNN have the advantage of parallelizable architecture that does not rely on sequential processing and less trainable parameters.

III. Tools & Hardware

The accompanying Python source code is available on `GitHub` under the MIT License. Neural network training was conducted using the *PyTorch* framework and logging capabilities of *Tensorboard* where leveraged. Makefile and a *YAML* configuration file assist in accessing functionalities. First the raw data is inspected, cleansed and transformed to a clean input for modeling. Subsequently training is conducted on the prepared dataset and validated with an independent test set. For evaluation a various indicators were tracked: Loss, Accuracy, Precision, Recall, F1-Score, training time and CPU & GPU utilization.

The hardware used for training and benchmarking is a Linux x86-64 system equipped with an i5-3450 CPU @ 4 x 3.10GHz, 16GB RAM and a 6GB GTX 1660 GPU.

IV. Dataset

The dataset [8] used for evaluation consists of around 10.000 patients with various cardiovascular diseases that were labeled by professional experts. Every recording originates from a 12-lead ECG with a sampling rate of 500Hz and a duration of 10 seconds. The authors offer an already denoised version of the raw ECG data, which is the chosen basis for this report.

A. Profiling

The set includes files with additional meta data with more information about the subjects and explanation of labels. The target variable for this set is denoted as the diagnosed rhythm of the corresponding patient. A first look at the label distribution (Figure 2) reveals 11 different categories (see Table I) , which are partly imbalanced. As this influences the classifier gravely a modification of the labels is done. To create an approximate balance the labels will be remapped: *Sinus Rhythm* (SR), *Sinus Bradycardia* (SB) and *Atrial Fibrillation* (AFIB) remain independent categories while SVT, AT, AVNRT and AVRT are mapped to *Sinus Tachycardia* (ST). The remaining labels are dropped. As it is shown in

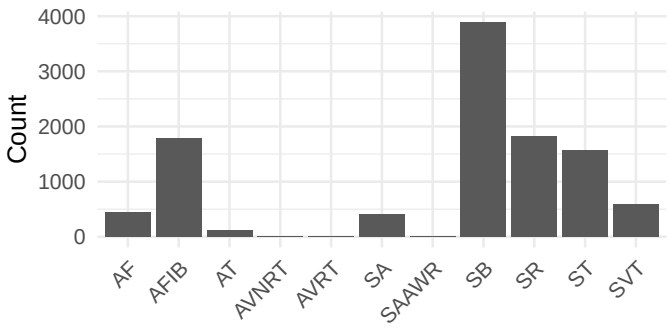


Fig. 2: Distribution of rhythm classification

Acronym Name	Full Name
SB	Sinus Bradycardia
SR	Sinus Rhythm
AFIB	Atrial Fibrillation
ST	Sinus Tachycardia
AF	Atrial Flutter
SVT	Supraventricular Tachycardia
AT	Atrial Tachycardia
AVNRT	Atrioventricular Node Reentrant Tachycardia
AVRT	Atrioventricular Reentrant Tachycardia
SAAWR	Sinus Atrium to Atrial Wandering Rhythm

TABLE I: Mapping of acronyms to full name

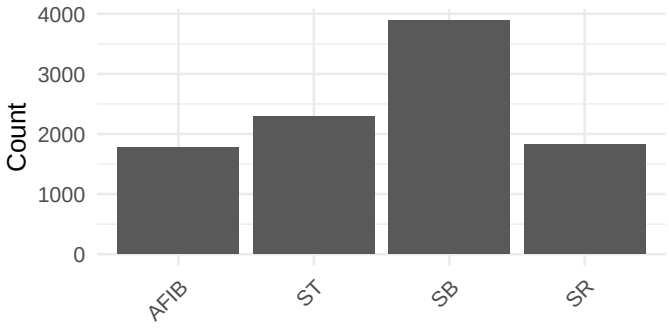


Fig. 3: Distribution of rhythm classification after reordering

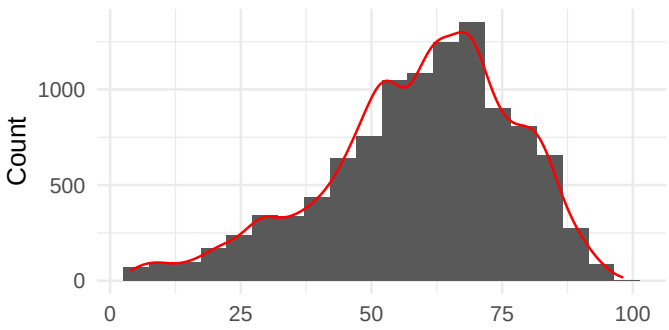


Fig. 4: Distribution of patient age

Figure 3 the set now consists only of four categories: SR (normal rhythm), SB (slower rhythm), ST (faster rhythm) and AFIB (irregular beating of the atrial chambers). SB is still dominant but the other categories appear well balanced. The participating patients age is distributed over all age groups but with a mean of ∼59 years has a prevalence in older age groups (Figure 4). The resulting density function therefore has a negative skew paired with a moderate kurtosis. A further view on the gender distribution (Figure 5) show slightly more male patients (55%). A boxplot (Figure 6) examines the correlation between diagnosis, gender and age. While the variance for both genders per diagnosis stays approximately constant, the mean of the female distributions seem to be lower than the male counterparts. Interestingly for SB and ST the mean is roughly equivalent to the global mean but when investigating AFIB it is significantly higher (∼75 years). This meta data is definitely useful information for a classifier but is neglected in this case as the focus is on ECG time series. The sensor data consists of over 10.000 files with 5000 rows and 12 columns respectively, each filled with real-valued numbers in millivolts (mv) unit. The 12 columns correspond to the layout of a standard 12-lead ECG: The first three columns are the limb leads (*I*, *II*, *III*), the next three columnmns are the augmented limb leads (*aVR*, *aVL*, *aVF*) and the remaining six depict the precordial leads (*V1*, *V2*, *V3*, *V4*, *V5*, *V6*). For visualization a complete sample for all leads across the 10 second recording span can be found in Appendix A.

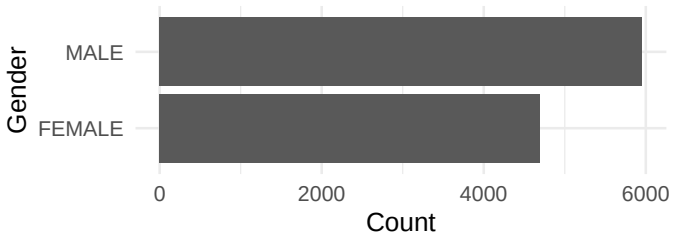


Fig. 5: Distribution of gender

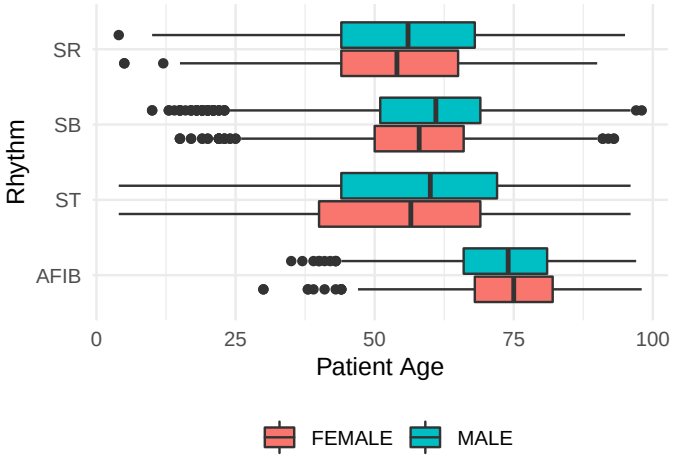


Fig. 6: Distribution of patient age per diagnosis and gender

B. Preprocessing

A big part of the preprocessing procedure is already done as the data is provided as a denoised version. When loading the files to memory all records seem to be complete but for one. A check for NaN-values reveal a percentage of 2.5% missing values. As a rule of thumb all records that are missing 20% of values are removed from the set. By checking the standard deviation of the records it is also revealed that some leads don't show any fluctuation. Therefore all records that contain leads with zero deviation are removed from the set as well. The last check concerns outlier values. At this point it has to be mentioned that domain knowledge to reliably define a threshold to separate outliers from anomalies is not available. Subsequently the threshold is set to 10.000 mv

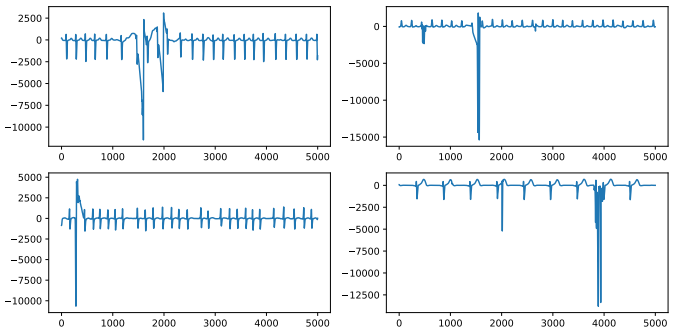


Fig. 7: Sampled leads with erroneous measurements

to be sure to only delete records with a high chance of being erroneous. A elaborated investigation could reveal more measurement mistakes. Examples of these critical rows are found in Figure 7. After eliminating suspicious rows the data set has no missing values left. The final dataset consists of records from 9.725 patients. Another aspect is the length of the time series with 5000 samples for 10 seconds of recording. Even when using an LSTM it can be problematic to propagate error over this distance. Generally speaking a sequence length of 250-500 is recommended for using recurrent architectures. With a sampling rate of 500Hz downsampling is a valid option without losing too much information. Further, as the pattern of a hearbeat is recurring in the record as well ($\sim$1-1.6 beats per second for a healthy person) the data is restricted to the first 5 seconds, which reduces the sequence length to 2500 in raw format. A crucial part of downsampling is to keep the amplitudes of the series as best as possible. The *Largest-Triangle-Three-Buckets* algorithm [9] promises to keep this requirement and therefore is chosen in this case. Next, the target length needs to be defined. A visualization of different downsampling thresholds is displayed in Appendix B. A threshold of 350 reduces the data points remarkably and keeps the majority of characteristics of the series.

The preprocessing requires to load a high number of files from disc followed by the single threaded downsampling algorithm. To accelerate the procedure and utilize all CPU-cores of the machine, multi-processing is applied. As expected the processing time scales linear to the number of workers. Using all 4 cores of the machine yield a speed-up from approximately 4 and a drop in processing time from around 40 min to 10 min. As more workers than available cores are spawned, the increase in performance stops immediately. On the contrary it takes even a few seconds longer, which is probably caused by concurrency for resources.

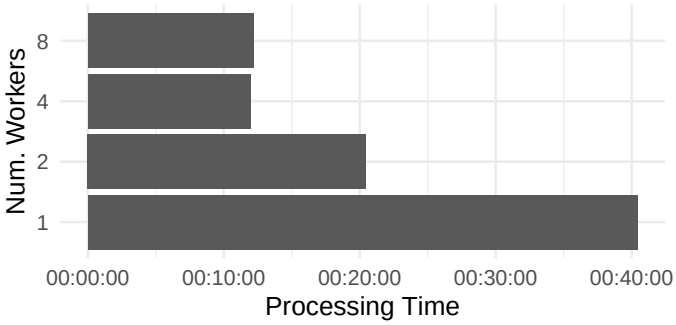


Fig. 8: Preprocessing time for various amounts of workers

Next, the dataset is parted into a training and test set via a stratified split to preserve the label distribution. For the experiments a split of 80% training and 20% validation is considered. Eventually the data is standardized by shifting with the mean and scaling to unit variance, which is a necessity for reliable training with neural networks. The resulting data is saved to a binary file that allows fast loading for the subsequent experiments.

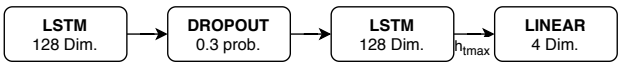


Fig. 9: LSTM Classification Architecture

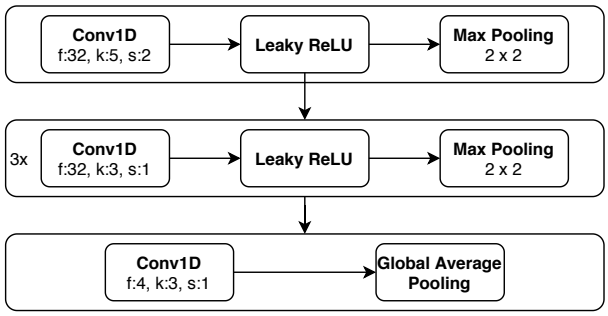


Fig. 10: CNN Classification Architecture

V. Experiments

For benchmarking a various experiments were conducted for both the LSTM and the CNN approach. Both have a basis architecture, where parameters are varied throughout the test runs. The standard LSTM architecture consists of 2 stacked units with a hidden dimension of 128 with a 0.3 dropout in between. The last hidden state of the second layer is then fed into a fully-connected layer to output class scores (Figure 9). Training time is 100 epochs. The standard CNN has 4 1D-convolutional blocks, each with a filter size of 32 with subsequent application of *Leaky ReLU* as non-linearity & *Max Pooling*. Eventually a fifth convolution in combination with *Global Average Pooling* yields the desired class scores (Figure 10). CNN training is conducted with 50 epochs. Globally the chosen optimizer is *Adam* [10] with a learning rate of 0.001 and the *cross-entropy* loss is used. Further, the batch size is 64. Due to constrained time and resources the experiments where only conducted once, which means no cross-validation was applied. Hence, no confidence intervals of the results are supplied and the obtained metrics are subject to uncertainty. Each experiment category contains a result table with the entries described in Table II. Additionally, the corresponding training history is attached in the appendix.

Metric	Description
Acc.	Validation Accuracy
Prec.	Validation Precision
Rec.	Validation Recall
F1	Validation F1-Score
# Ep.	Episode that yielded the best results
Ep. ⏲	Average ep. processing time in seconds
CPU	Average CPU utilization
GPU	Average GPU utilization

TABLE II: Mapping of acronyms to full name

A. LSTM: Batch Size

Batch size describes the number of samples that are fed into the model in parallel and used for one step of gradient descent. The higher the batch size is, the more likely it is to approximate the true direction of the gradient. This comes with the downside of high calculation effort and memory limitations. By choosing a small batch size theses issues can be mitigated but the result is a more noisy and stochastic gradient. On the other hand with a small batch size it may be possible to escape local minima and more batches in one epoch mean more weight updates per epoch, which is one reason for faster convergence. For the experiments the batch sizes 16, 32, 64, 128, 256 and 512 were used. Table III displays the test results and graphs of the every training progress are in Appendix C. The quality measure most widely used for arrhythmia detection is the F1-Score, which is around 95.5% in most cases. The result confirmes the theory that smaller batch sizes give faster convergence. Further, a smaller batch (cf. 16, 64) yields a more unstable and fluctuating training progress, while the updates for the bigger batches are more holistic and consistent. Additionally, partly caused by the increased frequency of batch updates the training for smaller batches show early signs of overfitting, which is expressed by the increasing generalization gap between training and test metrics, especially in the loss function. Validation loss is increasing as the model gets more and more confident in the wrong decisions.

On the performance side the training time per epoch decreases almost linearly while the batch size increases. This goes in hand with rising utilization of the GPU and as the GPU arrives at maximum capacity (batch size 256 to 512) the gain in processing time stops. A batch size of 16 utilizes the GPUs power by only 50%. The CPU load stays consistent as its task is mostly tracking metrics because the dataset is already loaded to memory and is being served to the GPU via non-computational intensive index operations. Overall a batch size of 128 seems to be a good fit as it yields not only the best F1-Score but also keeps the balance between most aspects of choosing a batch size: High system load and fast convergence.

Batch Size	Acc.	Prec.	Rec.	F1	# Ep.	Ep. ⌀	CPU	GPU
16	96.04	95.65	95.45	95.55	86	21.86	37.96	49.08
32	95.78	95.37	95.1	95.23	69	11.66	38.79	53.86
64	95.58	95.16	94.81	94.96	87	6.11	39.13	69.61
128	96.04	95.55	95.64	**95.57**	72	3.47	46.99	84.32
256	95.99	95.55	95.31	95.42	96	2.69	43.1	90.66
512	93.98	93.77	92.63	93.12	94	2.36	41.06	94.85

TABLE III: LSTM results for diff. batch sizes

B. LSTM: Hidden Dimension

The next experiment concerns with varying the dimensions of the hidden states. An increase of dimension yields an increase in trainable parameters, which also increases system load and training time. On the other side this also extends the hypothesis space of the model, which enables a fitting of more complex functions. The downside is an affinity for overfitting, when the training data is rather small. The setup in this case are the hidden dimension sizes of 16 (4.1k parameters), 32 (14k param.), 64 (52k param.), 128 (205k param.), 256 (803k param.) and 512 (3.2 mio param.). The parameter size scales quadratically due to the internal structure of a LSTM (cf. Section II). The training progress (Appendix D) indeed reveals a tendency to overfit with growing dimensionality. On the contrary even the smallest model with only 0.13% the parameters of the biggest model is able to achieve an F1-Score of 88.5% (Table IV). While in this case the hypothesis space might be too restricted, the dimensions of 32 and 64 also have a rather moderate parameter complexity and yield F1-Scores of 92.4% and 95% respectively.

A growing number of trainable parameters usually comes with more stress on the system. With increasing dimensionality the GPU utilization is also steadily increasing. Interestingly the time per epoch experiences only a marginal increase until the maximum GPU utilization is met, which then nearly triples it (8 sec. to 22 sec.). The reason for this is the capability of *CUDA cores* to process a lot of data in parallel with the same operator but when reaching capacity limits it still has to be queued, which results in the drastic runtime increase.

Hidden Dim	Acc.	Prec.	Rec.	F1	# Ep.	Ep. ⌀	CPU	GPU
16	89.92	88.72	88.44	88.55	100	5.46	31.76	37.86
32	93.42	92.81	92.09	92.41	100	5.3	38.59	42.32
64	95.63	94.99	95.21	95.08	86	6.16	39.06	42.61
128	95.58	95.16	94.81	94.96	87	6.11	39.13	69.61
256	95.89	95.32	95.37	**95.34**	93	8.56	45.24	88.95
512	94.6	94.14	93.73	93.86	94	21.95	43.01	95.71

TABLE IV: LSTM results for diff. hidden dimensions

C. LSTM: Stacked Layers

Another technique to boost expressive power is by stacking layers to introduce another degree of non-linearity. Similar to increasing the hidden dimension adding more layers increases capability and trainable parameters. In this experiment the examined stacked layers are 1 (73k param.), 2 (205k param.), 3 (337k param.) and 4 (470k param.). The increase in parameters is scaling linearly. The results in Appendix E reflect the theoretical considerations: More layers require more training to converge and are more prone to vanishing or exploding gradients. While LSTM help to mitigate this issue, the problem is still existent. 1-2 layer already begin to overfit after 60 epochs, while 3-4 layer might benefit from even longer training. Further, increasing the number of layers comes with a steady increase of F1-Score (Table V), while the system load stays rather consistent. This is due to the sequential nature of recurrent networks and also most other architectures, that processes layers piecewise. Instead the epoch time increases linearly around 3 seconds with every layer added. Overall, while choosing more layers seems to equal a better testing performance, the potential to overfit also increases including growing training times.

Layers	Acc.	Prec.	Rec.	F1	# Ep.	Ep. ⏲	CPU	GPU
1	94.4	93.9	93.41	93.63	92	3.14	42.22	55.35
2	95.58	95.16	94.81	94.96	87	6.11	39.13	69.61
3	95.78	95.42	95.15	95.26	99	8.69	44.04	74.78
4	96.4	96.17	95.85	**95.99**	86	11.99	38.32	71.21

TABLE V: LSTM results for diff. stacked layers

D. LSTM: Dropout

Dropout [11] is a common regularization practice in the domain of deep learning. A dropout layer induces a probability of zeroing elements in the respective tensors. By doing so the network learns not to rely on specific features and therefore should generalize better. A side effect are larger weights in the model, which is mitigated by multiplication with the dropout probability during inference. This experiment tests different dropout rates after the 2 LSTM layers starting from 0 up to 0.9 zeroing probability. The training progress (Appendix F) shows that with growing dropout rate the signs of overfitting vanishes. A dropout of 0.3 seems as a good fit to make training exceedingly stable. A rate of 0.5 delivers the best F1-Score of 95.55% (Table VI) but the fitting looks less consistent. By increasing the probability the training deteriorates more and more, which is a logical response if only a fraction of the original data is available. Interestingly even with a dropout of 90% the LSTM is capable of learning important concepts and achieves an F1-Score of 92.58%. One could argue, that based on the recurrent nature of the ECG necessary information for the classification could be found in nearly every state h_t. On the side of system load there is no relevant difference in the metrics, as zeroing elements is an efficient operation. Only not using any dropout layers makes it around 0.6 seconds faster.

Dropout	Acc.	Prec.	Rec.	F1	# Ep.	Ep. ⏲	CPU	GPU
0	95.37	94.9	94.89	94.86	69	5.64	37.09	64.5
0.1	96.09	95.91	95.36	95.61	89	6.3	32.96	65.29
0.3	95.58	95.16	94.81	94.96	87	6.11	39.13	69.61
0.5	95.99	95.63	95.49	**95.55**	100	6.22	40.37	66.43
0.7	95.37	95.24	94.48	94.78	99	6.16	42.09	67.89
0.9	93.37	92.73	92.63	92.58	96	6.23	38.39	66.54

TABLE VI: LSTM results for diff. dropout rates

E. CNN: Batch Size

After evaluating the LSTM in various categories similar experiments are conducted employing the previously described CNN to solve the same classification task with matching data. The first test analogously examines the effect of different batch sizes. The first remarkable aspect of the CNN is the extremely fast and stable convergence in all cases (Appendix G). While a small batch size of 16 and 32 is prone to early overfitting, it also gives the best results with a F1-Score of over 95% (Table VII). The larger batch sizes might benefit from longer training, as the best round indicates. Regarding processing times the CNN is getting increasingly faster with growing batch size, as computation can be handled in parallel. A batch of 512 yields an epoch time of 0.45 seconds, which implies that the network has breached the 90% accuracy mark after 3 seconds of training. This also comes only with a marginal increase of GPU utilization. Even for the largest batch size the GPU load is only 26.6%. For the first time a definite increase of CPU usage is recorded due to advanced speed of the epoch time. In summary as in this case memory is not a limiting factor it might be beneficial to stick to larger batches.

Batch Size	Acc.	Prec.	Rec.	F1	# Ep.	Ep. ⏲	CPU	GPU
16	95.84	95.87	94.79	95.28	27	3.76	47.59	17.06
32	95.99	95.74	95.22	**95.47**	20	1.98	44.98	23.7
64	95.37	94.98	94.55	94.76	50	1.03	48.38	21.8
128	95.58	95.0	94.9	94.94	36	0.69	54.48	23.7
256	95.12	94.69	94.27	94.46	48	0.51	60.95	31.82
512	94.5	94.32	93.41	93.78	50	0.45	52.48	26.6

TABLE VII: CNN results for diff. batch sizes

F. CNN: Number of Filters

The filter size of the LSTM is the equivalent to the hidden dimension of the LSTM. This experiment is conducted with varying filters with sizes 8 (1k param.), 16 (3.5k param.), 32 (11k param.), 64 (42k param.), 128 (157k param.) and 256 (609k param.) in each layer. Every filter comes with its own weight set and its number of parameters are dependent on the kernel size and the dimension of the input, which is the reason for the polynomial scaling. More filters conclude in more expressive power but also the capability to overfit if not enough training data is supplied. Appendix H shows exactly this behaviour. A model with 8 filters is limited in its hypothesis space but still achieves respectable results (Table VIII). On the other hand a high amount of filters also achieved the best F1-Scores of up to 95.99%. Here it could be argued whether the marginal increase in test score justifies the increased parameters and system load. Interestingly the best epochs of smaller filter sizes are clearly later than the ones of their counterparts. This could also be reasoned by the aspect of overfitting in high capacity models. The epoch processing time stays constant as the GPU load steadily climbs, which is due to parallelization.

# Filters	Acc.	Prec.	Rec.	F1	# Ep.	Ep. ⏲	CPU	GPU
8	93.93	93.42	92.64	93.01	41	1.02	49.5	21.66
16	95.42	95.01	94.59	94.78	48	1.04	47.38	19.5
32	95.78	95.5	94.9	95.17	37	1.03	48.69	23.1
64	95.94	95.52	95.44	95.48	41	1.02	49.81	26.98
128	96.4	95.89	96.1	**95.99**	16	1.04	50.47	42.72
256	96.3	96.02	95.65	95.82	29	1.23	42.61	63.36

TABLE VIII: CNN results for diff. number of filters

G. CNN: Stacked Layer

By adding more layers in an CNN the degree of abstraction and non-linearity increases as well. The tests are run with layers ranging from 1 with only 244 parameters up to 6 with 15k parameters. The simplest model consists only of the decision layer and therefore is rather limited in learning concepts. Indeed the training progress in Appendix I shows severe signs of underfitting with a high bias component. The model still performs double as good as random guessing, which still makes it a strong learner. Increasing the number of layers come in hand with a steady increase in F1-Score (Table IX) but as also previous experiments showed an affinity to overfit. It should be mentioned that stacking CNN layers with *valid* padding results in the transformed sequence becoming shorter with every step, which makes 6 layers nearly the maximum in this experiment. Using *same* padding would be an option to prevent this from happening. The system load stays constant and time for an epoch increases around 0.1 seconds on average as the stacked layers are processed sequentially. Overall, having 5 layers seems like a good fit for this scenario. At this point the increase in performance stagnates and even the best F1-Score is obtained, which should again be confirmed by cross-validation.

# Layers	Acc.	Prec.	Rec.	F1	# Ep.	Ep. ⏲	CPU	GPU
1	65.35	58.91	56.25	53.56	39	0.6	53.64	20.0
2	79.02	77.56	75.29	76.19	50	0.7	52.3	21.42
3	89.51	88.56	88.13	88.34	37	0.84	48.89	21.42
4	93.52	93.15	92.42	92.67	49	0.93	49.39	22.24
5	95.53	94.96	94.87	**94.92**	37	1.05	46.27	19.96
6	95.37	94.85	94.67	94.76	46	1.12	47.17	21.4

TABLE IX: CNN results for diff. stacked layers

VI. Discussion

The fundamental research question in this paper concerns with whether to trust on recurrent or convolutional approaches when classifying ECG time series. The LSTM is exactly designed to handle this kind of data and is able to express temporal dependence. The CNN on the other hand, as it was originally developed to cope with computer vision tasks, does not have this ability but its main strength is to identify relevant patterns in the data. On the side of trainable parameters the CNN has significantly less parameters due to the weight sharing property. Against this the LSTM has multiple fully-connected layers and pointwise multiplications in its internals, which makes it more heavy-weight and computationally intensive to train. In fact, the baseline CNN has not a single fully-connected layer equipped as *Global Average Pooling* is used to create class scores. This also helps to regularize training, prevent overfitting and is an efficient operation. The experiments show that the CNN puts way less stress on the system, converges in a fraction of time and achieves the same results. Both algorithms had a maximum F1-Score of exactly 95.99%. Further, convolution can be handled extremely efficient on GPU, while the sequential processing of recurrent units slows down the procedure. The increased number of parameters in an LSTM also contribute to more memory usage and nearly maximum GPU utilization in edge cases whereas the CNN experiments barely reached the 65% mark.

An additional aspect is the significant stable, robust and consistent training progress of the CNN, which is in contrast to some deterioration that happened during LSTM training in certain settings that could be due to high variance and high-dimensional non-convex error surfaces. A reason for the success of the CNN could be the translation-invariance property as the model only finds patterns and is detached from temporal dependencies. Further, due to convolution the input length is reduced layer by layer. This could imply that only relevant information in lower dimensional space is kept, which helps to extract crucial patterns. On the downside the CNN relies on a fixed input size whereas the LSTM can handle dynamic inputs. Additionally it has to be stated that this analysis might only be relevant in this exact use case. LSTMs have a wide variety of application and can also serve in sequence generation or sequence-to-sequence tasks. For a final opinion further tests and experiments are necessary but as this snapshot indicates one could benefit by utilizing CNNs for medical time series.

VII. Future Work

The proposed paper only concerned with the characteristics of ECG time series. For this dataset more meta-data is available and as the preprocessing shows it holds correlation to specific class labels. Therefore utilizing these as well could resolve in higher scores and more reliability in classification. The authors of the datataset [8] achieved a maximum of 96.5% by using this additional information with manually handcrafted time series attributes and gradient boosting. By relying on representation learning via CNN/LSTM and adding meta-features by a fully-connected layer this benchmark might be achievable with further tuning and experiments. Additionally the model could benefit from a hybrid model where a LSTM transforms the time series first and afterwards a CNN is utilized to extract information out of the high-dimensional hidden states.

As basis the whole process could benefit from a more sophisticated preprocessing with additional domain knowledge. Also for these experiments the ECG sequence was only used to 50% and then downsampled. More experiments could examine how feeding raw data to a model impacts testing performance and runtime.

Another variant could concern more experiments using a *Gated Recurrent Unit* (GRU) instead of an LSTM, which is less complex and therefore faster to train with increased ability to cope with small training data. Eventually, activations of the respective layers could indicate where critical areas for the classification are located in the time series, which can be visualized and then be consulted for further steps.

VIII. Conclusion

This project examined the capabilities of LSTM and CNN approaches for detecting arrhythmias in computerized ECGs. Experiments have shown that both algorithms are able to produce similar test results. Given the accompanying dataset modern deep learning is able to identify a heart illness with a confidence of over 96% by only using ECGs without metadata. More tests and cross-validation have to be conducted to confirm the opinions made in this paper. In summary a CNN converges faster, trains more stable and puts less stress on the computing system in comparison to a LSTM.

References

[1] J. Zheng, H. Chu, D. Struppa, J. Zhang, S. M. Yacoub, H. El-Askary, A. Chang, L. Ehwerhemuepha, I. Abudayyeh, A. Barrett, G. Fu, H. Yao, D. Li, H. Guo, and C. Rakovski, "Optimal multi-stage arrhythmia classification approach," *Scientific Reports*, vol. 10, no. 1, p. 2898, 2020.

[2] A. Y. Hannun, P. Rajpurkar, M. Haghpanahi, G. H. Tison, C. Bourn, M. P. Turakhia, and A. Y. Ng, "Cardiologist-level arrhythmia detection and classification in ambulatory electrocardiograms using a deep neural network," *Nature Medicine*, vol. 25, no. 1, pp. 65–69, 2019.

[3] S. S. Xu, M.-W. Mak, and C.-C. Cheung, "Towards end-to-end ecg classification with raw signal extraction and deep neural networks," *IEEE Journal of Biomedical and Health Informatics*, vol. 23, no. 4, pp. 1574–1584, 2019.

[4] S. L. Oh, E. Y. Ng, R. San Tan, and U. R. Acharya, "Automated diagnosis of arrhythmia using combination of cnn and lstm techniques with variable length heart beats," *Computers in Biology and Medicine*, vol. 102, pp. 278–287, 2018.

[5] F. Lüer, D. Mautz, and C. Böhm, "Anomaly detection in time series using generative adversarial networks," in *2019 International Conference on Data Mining Workshops (ICDMW)*, IEEE, 2019, pp. 1047–1048.

[6] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[7] C. Olah. (2015). Understanding lstm networks, [Online]. Available: http://colah.github.io/posts/2015-08-Understanding-LSTMs/.

[8] J. Zheng, C. Rakovski, S. Danioko, J. Zhang, H. Yao, and G. Hangyuan, *A 12-lead electrocardiogram database for arrhythmia research covering more than 10,000 patients*, 2019. [Online]. Available: https://figshare.com/collections/ChapmanECG/4560497/2.

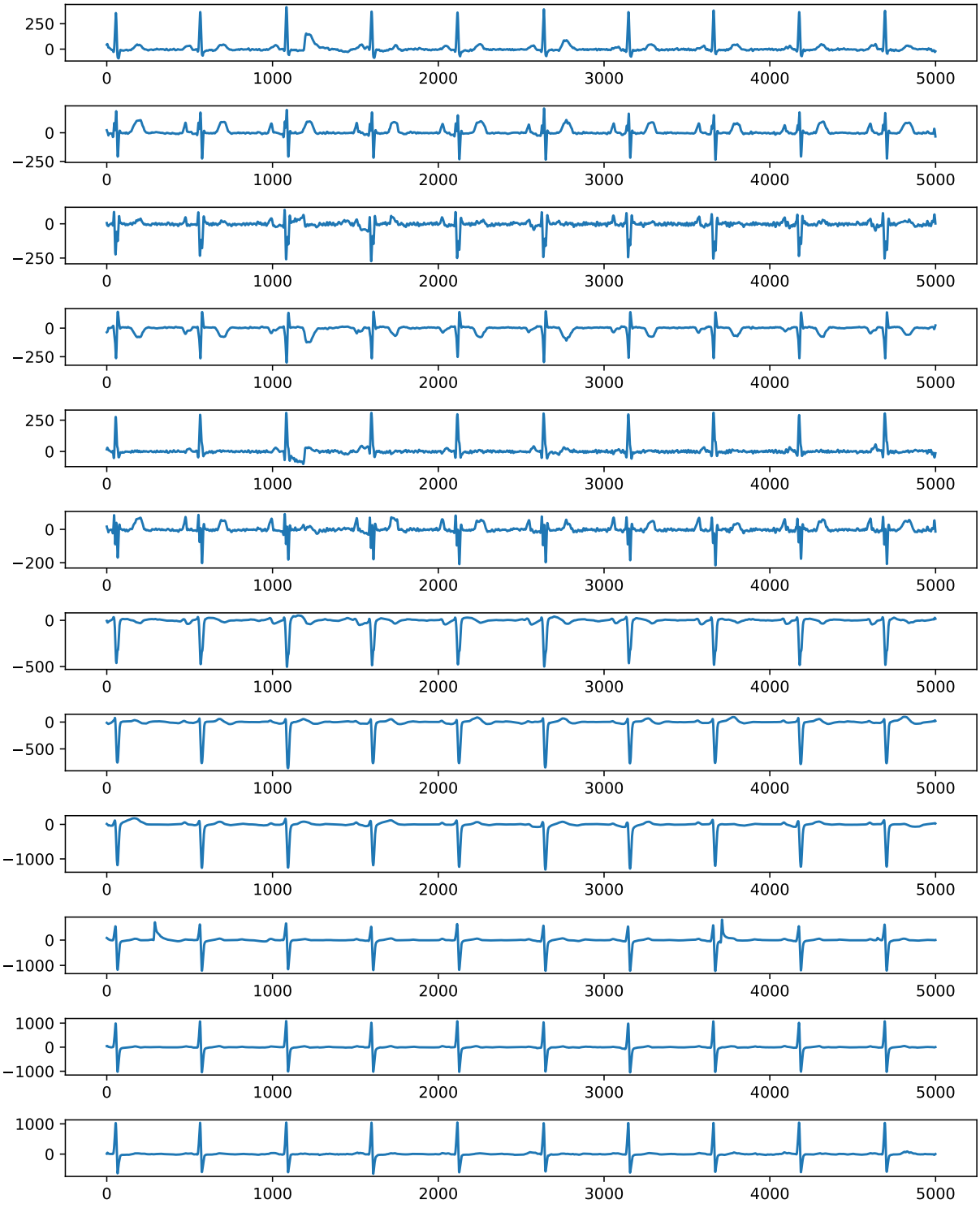
[9] S. Steinarsson, "Downsampling time series for visual representation," PhD thesis, 2013.

[10] D. P. Kingma and J. Ba, *Adam: A method for stochastic optimization*, 2014. arXiv: 1412.6980 [cs.LG].

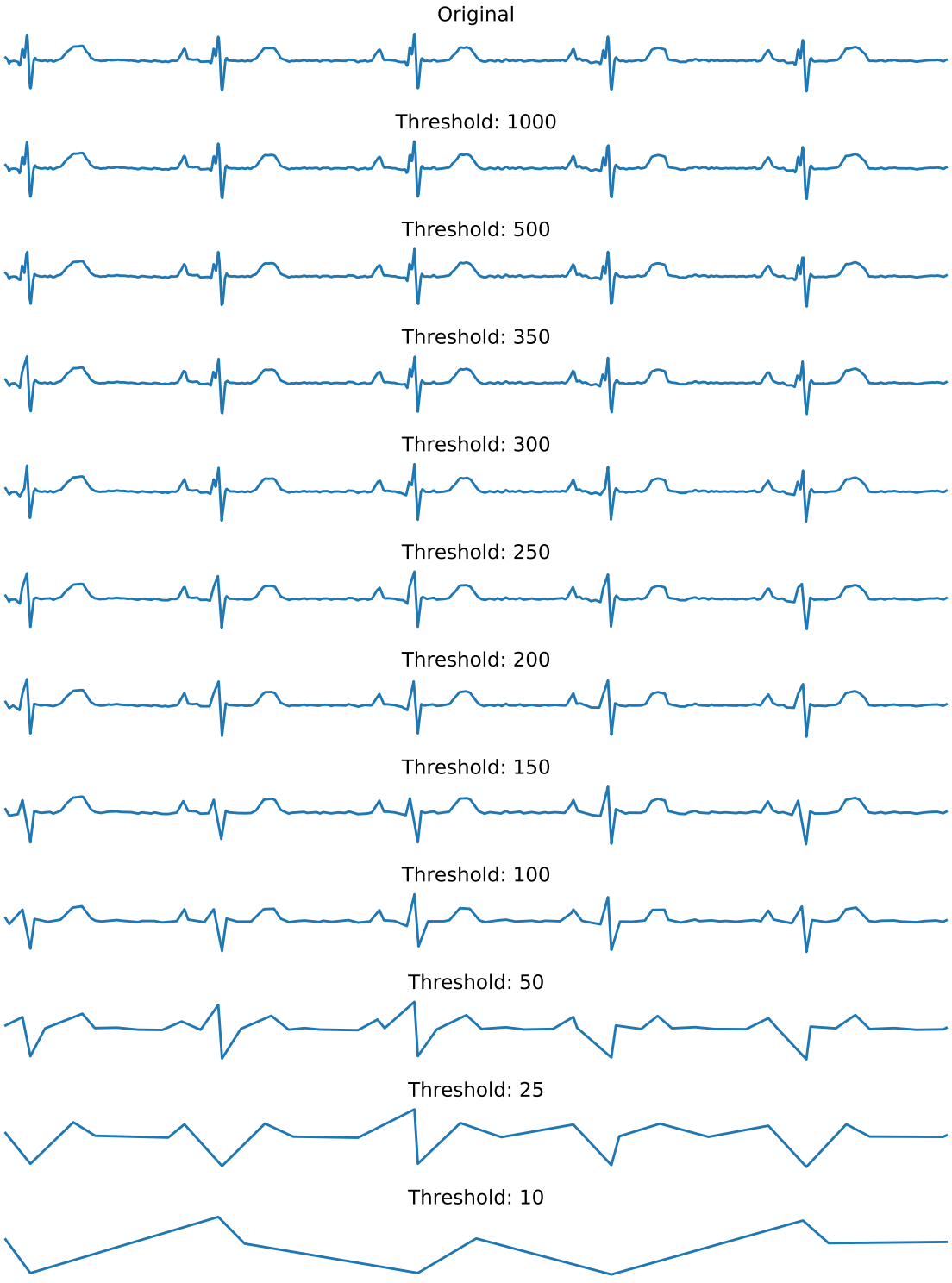
[11] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *The journal of machine learning research*, vol. 15, no. 1, pp. 1929–1958, 2014.

APPENDIX

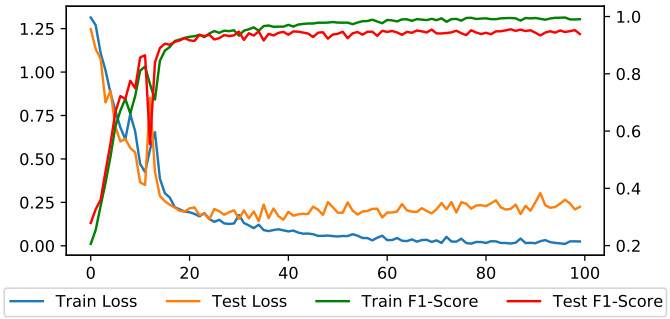
A. Plot of a 12-lead ECG sampled from the data set



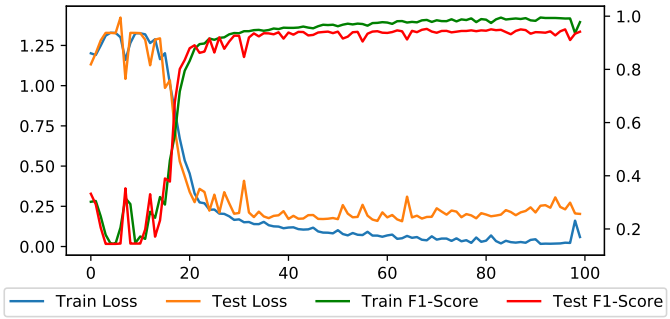
B. Plot of a 2500 step time series with different downsampling lengths



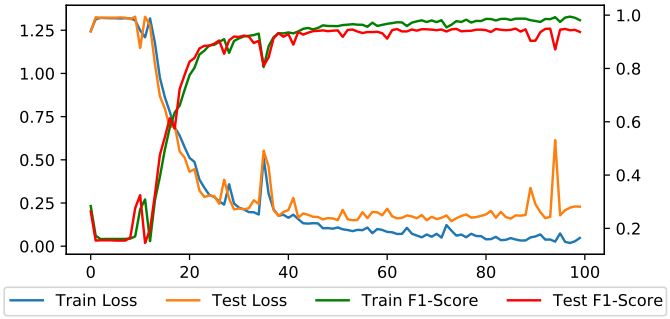
C. LSTM training progress for varying batch sizes



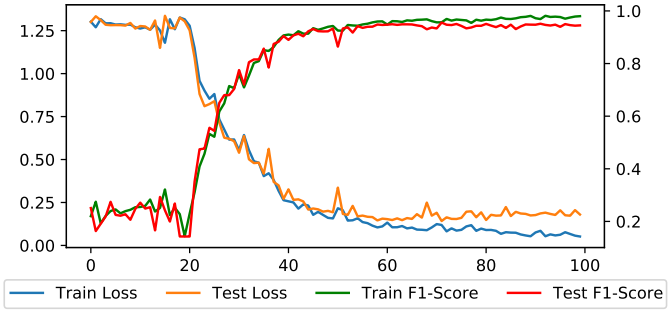
Batch size: 16



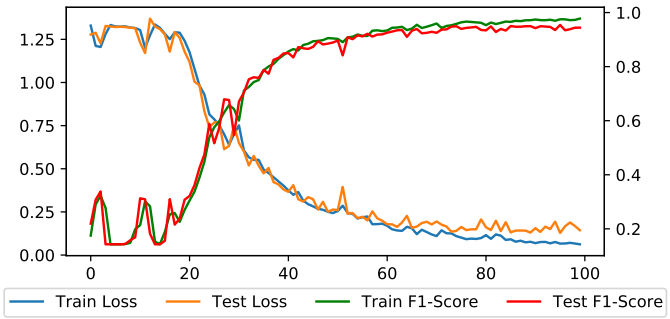
Batch size: 32



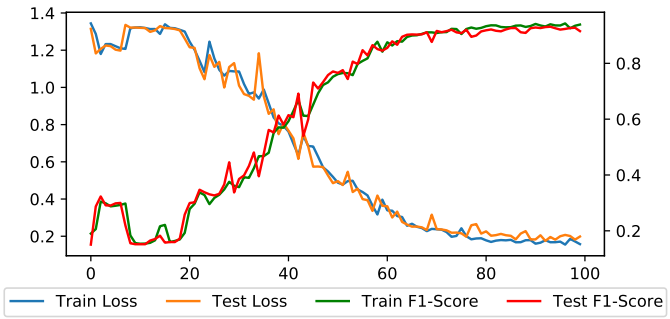
Batch size: 64



Batch size: 128

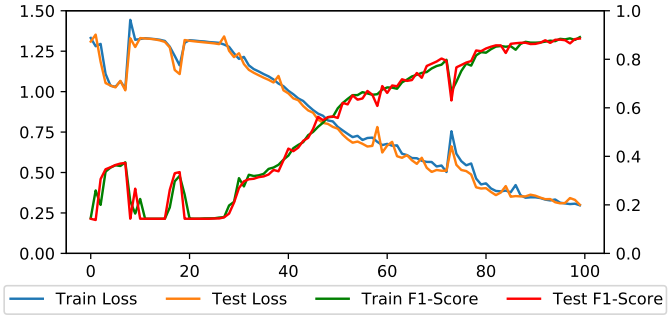


Batch size: 256

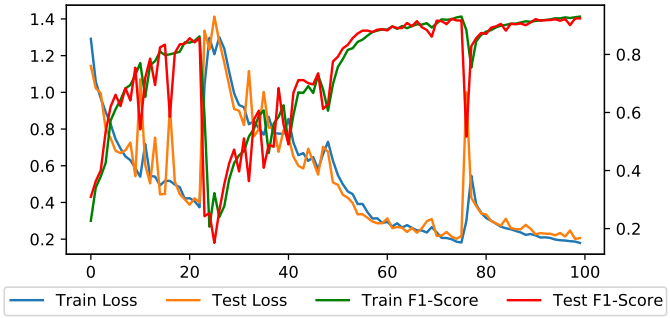


Batch size: 512

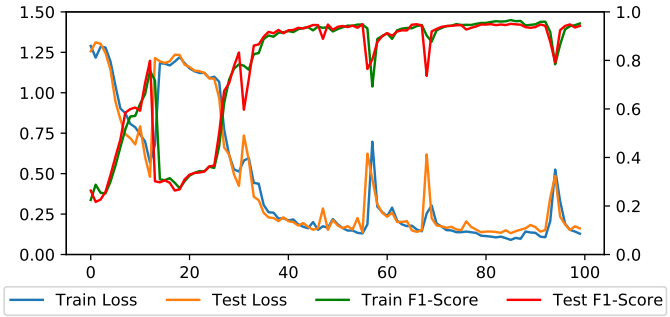
D. LSTM training progress for varying hidden dimensions



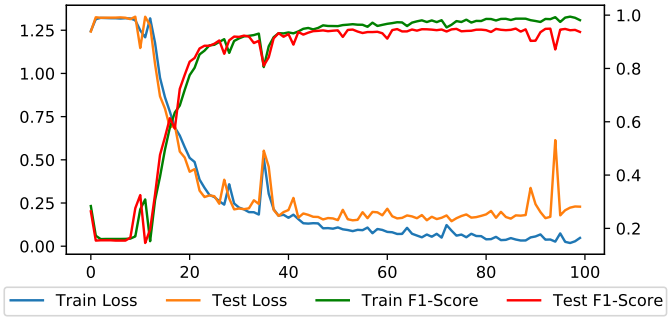
Hidden dimension: 16



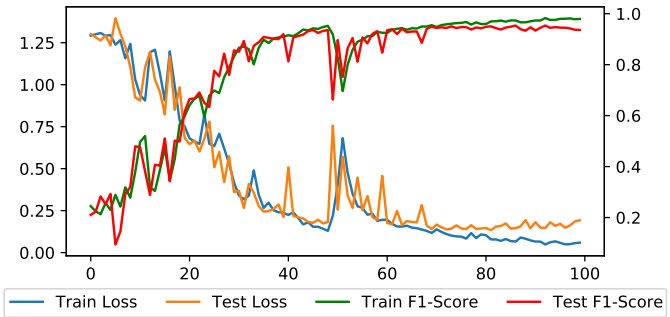
Hidden dimension: 32



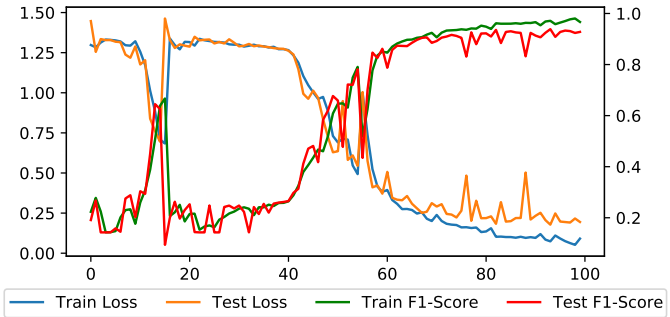
Hidden dimension: 64



Hidden dimension: 128

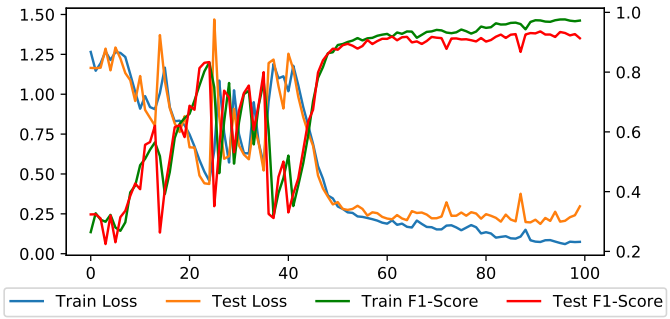


Hidden dimension: 256

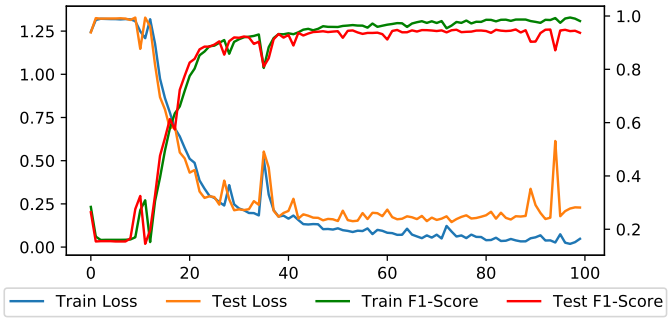


Hidden dimension: 512

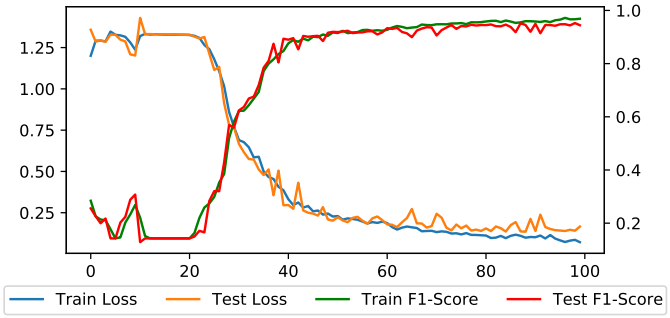
E. LSTM training progress for varying stacked layers



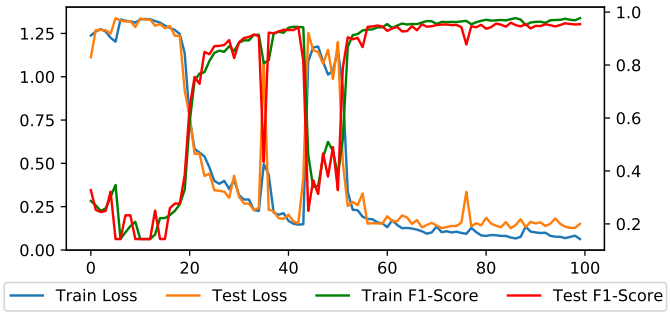
Stacked Layer: 1



Stacked Layer: 2

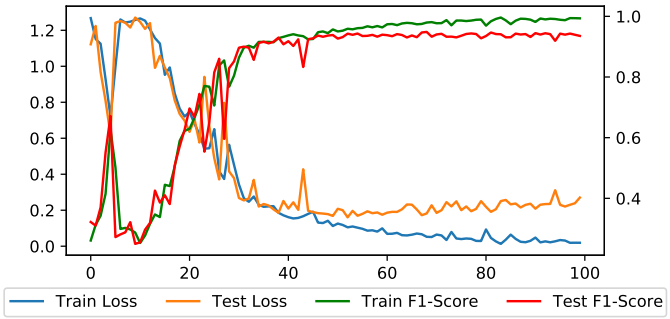


Stacked Layer: 3

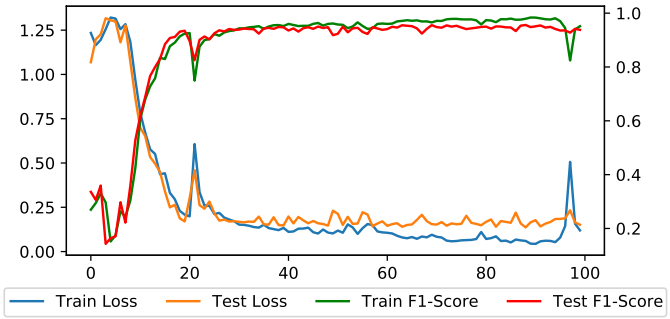


Stacked Layer: 4

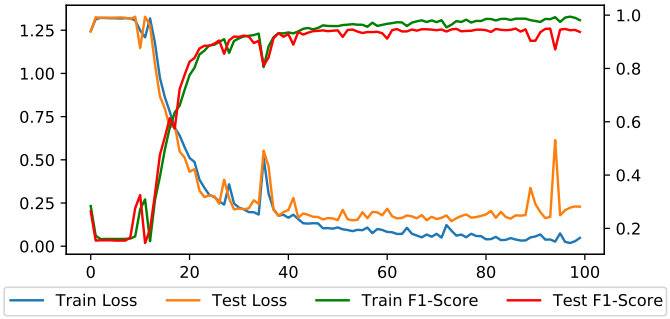
F. LSTM training progress for varying dropout rates



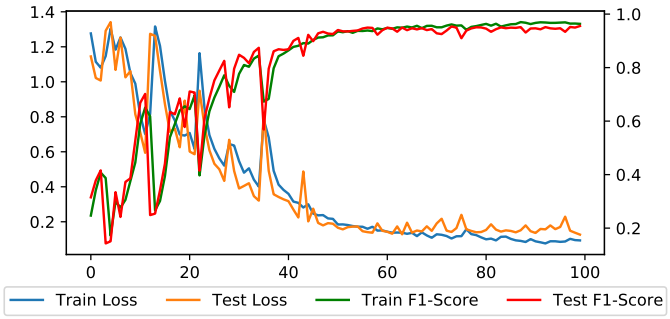
Dropout rate: 0



Dropout rate: 0.1

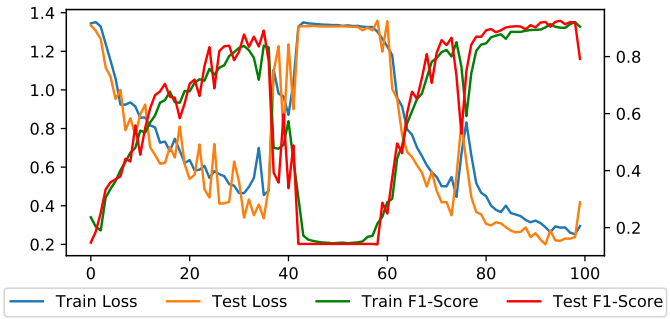


Dropout rate: 0.3



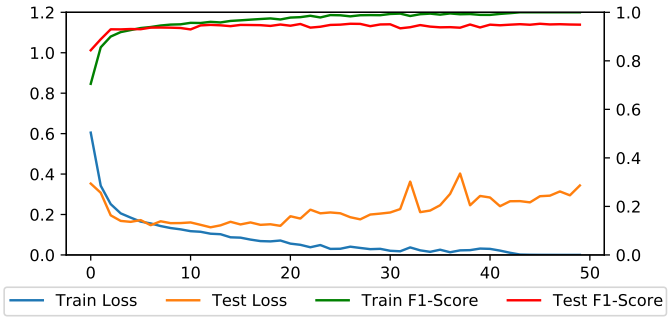
Dropout rate: 0.5

Dropout rate: 0.7

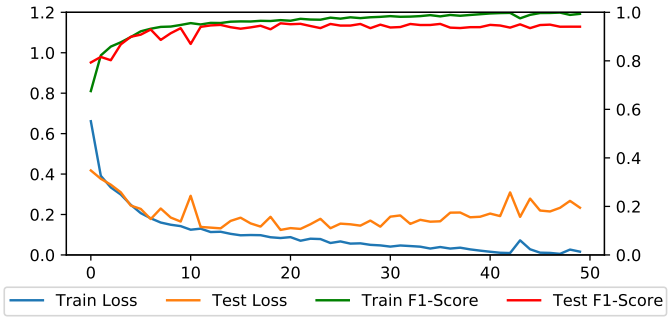


Dropout rate: 0.9

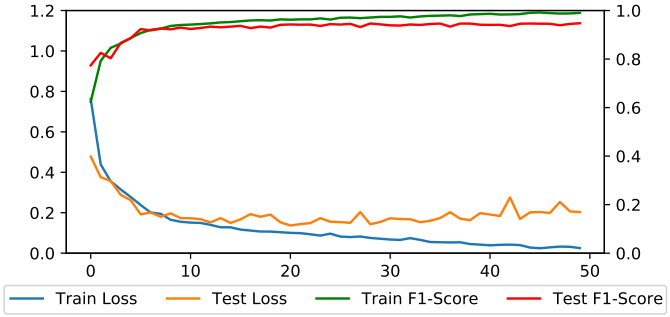
G. CNN training progress for varying batch sizes



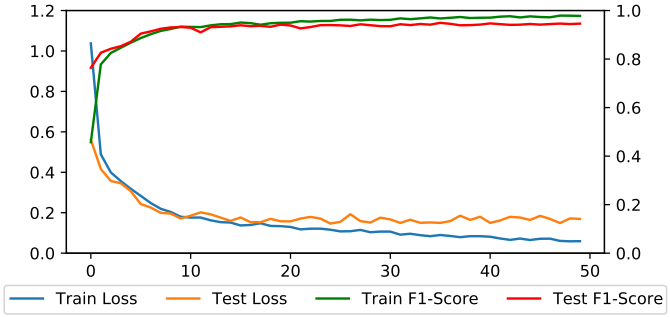
Batch size: 16



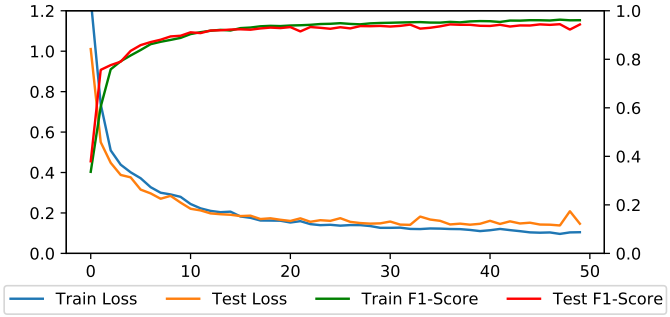
Batch size: 32



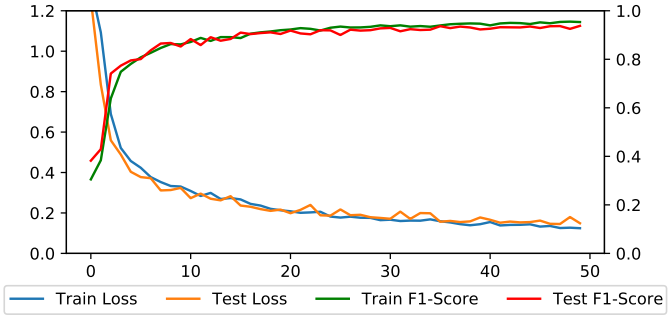
Batch size: 64



Batch size: 128

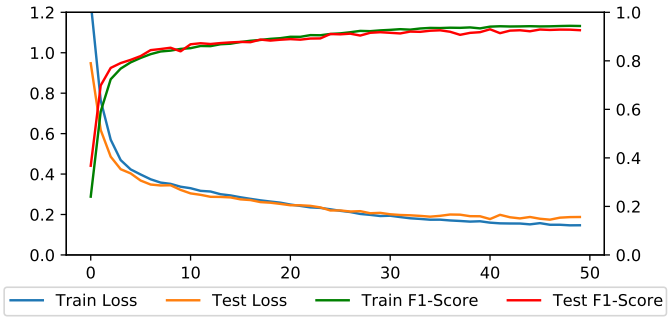


Batch size: 256

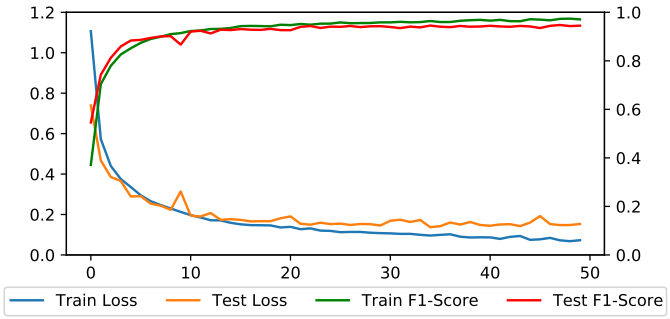


Batch size: 512

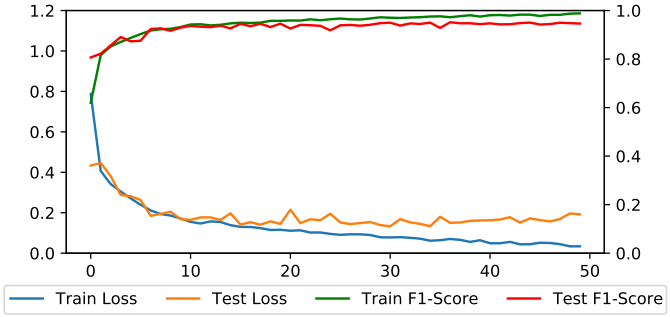
H. CNN training progress for varying number of filters



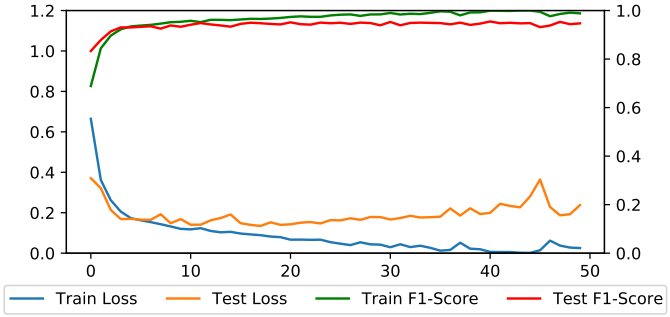
Num. filter: 8



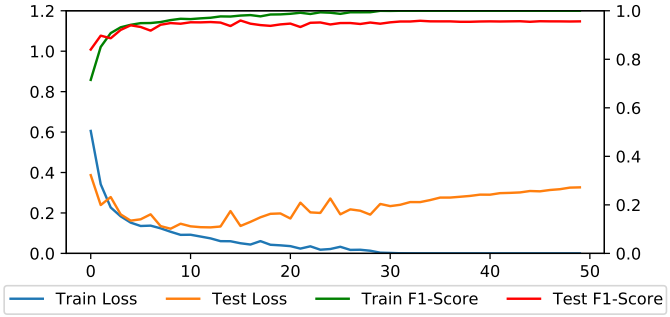
Num. filter: 16



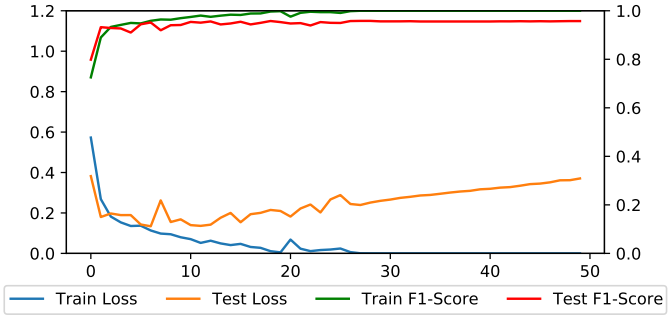
Num. filter: 32



Num. filter: 64

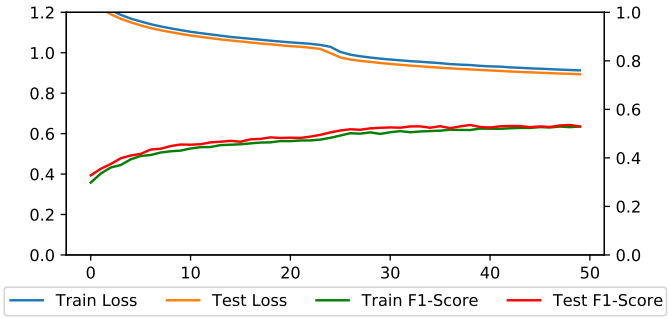


Num. filter: 128



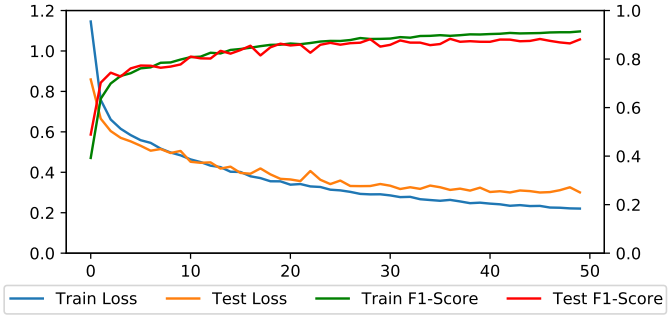
Num. filter: 256

I. CNN training progress for varying number of layer

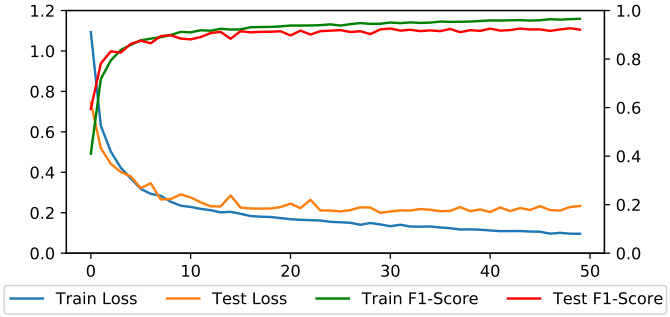


Num. layer: 1

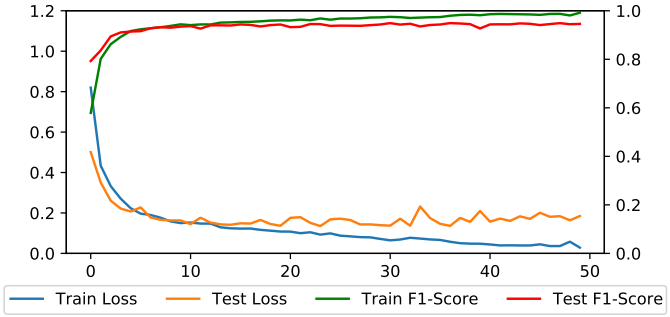
Num. layer: 2



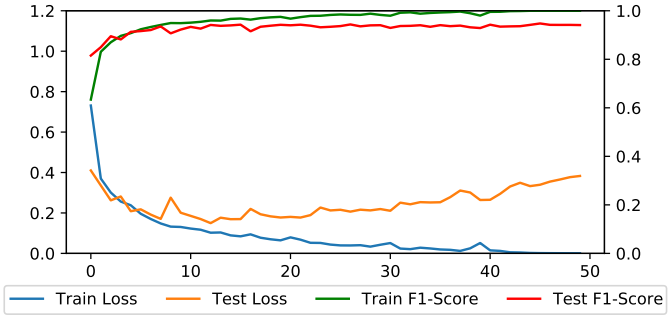
Num. layer: 3



Num. layer: 4



Num. layer: 5



Num. layer: 6