RESEARCH

Learning to detect the onset of slow activity after a generalized tonic-clonic seizure

Carroll Vance[1*†], Yejin Kim[2], Xiaoqian Jiang[2], Guoqiang Zhang[3] and Samden Lhatoo[3]

*Correspondence: cs.vance@icloud.com
[1]University of Houston, Houston, USA
Full list of author information is available at the end of the article
†Equal contributor

Abstract

Methods: In recent years, deep learning has become state of the art for many domains with large amounts data. Although healthcare has accumulated a lot of data, they are often not abundant enough for subpopulation studies where deep learning could be beneficial. Taking these limitations into account, we present a framework to apply deep learning to the detection of the onset of slow activity after a generalized tonic-clonic seizure, as well as other EEG signal detection problems exhibiting data paucity.

Results: We conducted ten training runs for our full method and seven model variants, statistically demonstrating the impact of each technique used in our framework with a high degree of confidence.

Conclusions: Our findings point toward deep learning being a viable method for detection of the onset of slow activity provided approperiate regularization is performed.

Keywords: electroencephalogram; sudden death in epilepsy; generalized tonic-clonic seizure; onset of slow activity; signal detection; machine learning; deep learning; neural network; convolutional neural network; data paucity

Introduction

Sudden death in epilepsy (SUDEP) is a rare disease in US, however, they account for 8–17% of deaths in people with epilepsy. This disease involves complicated physiological patterns and it is still not clear what are the physio-/bio-makers that can be used as an indicator to predict SUDEP so that care providers can intervene and treat patients in a timely manner. For this sake, UTHealth School of Biomedical Informatics (SBMI) organized a machine learning Hackathon to call for advanced solutions `https://sbmi.uth.edu/hackathon/archive/sept19.htm`.

Recent advancements deep learning have significantly improved performance for classification and detection tasks [1, 2]. However, generalization ability is still limited due to the lack of sufficient high-quality training data for many domains. This holds true for many problems in the biomedical domain where data is often limited (especially for sub-population studies), which constraints the capacity of highly powerful supervised deep learning frameworks [3]. Since deep learning is known for requiring a considerable amount of data [4], applying it to a problem such as detection of markers (onset of slow activity) to predict critical patterns in a rare disease like SUDEP is not straightforward.

Contributions

Our method attempts to build a framework to apply recent advancements in deep learning [5, 2, 6, 7] to detection problems such as detection of the onset of slow activity after a generalized tonic-clonic seizure, where availability of of training data is limited. We combine a variety of preprocessing (Resampling), regularization (Anti-aliased temporal downsampling [6], Global temporal downsampling [8], Global batch-wise z-scoring, Kernel regularization [9]), and optimization (Batch size [10], Loss discount factor) techniques to work around the data paucity issue. We also develop a system for real-time visualization of our models predictions to emphasize which parts of the signal contributed most to the decision https://www.youtube.com/watch?v=cDuRsh2pSRM.

Method Overview

From a high level, we feed an EEG sequence x into our binary classification model $y = f(x)$, which estimates the probability $y \approx P(y|x)$ that the sequence contains the onset of slow activity (i.e., label). The chosen model architecture is a residual neural network [11] utilizing stacked convolution layers [12], skip connections [11], batch normalization [7], downsampling [6], and non-linear activation functions. We train our model using mini-batch stochastic gradient descent (SGD).

Data

The original source of the training data D contains variable length sequences composed of recordings from ten pairwise offsets of two adjacent EEG electrodes [13]:

$$\{fp_1 - f_7, f_7 - t_7, t_7 - p_7, p_7 - o_1, fp_2 - f_8, f_8 - t_8, t_8 - p_8, p_8 - o_2, fz - cz, cz - pz\} \in F$$

The sequences were recorded from 134 different patients, each with their own variable length sequence [13]. It follows that $|D| = 134$. The EEG sampling rate F_s is 200 Hz, and each timestep t_n is labeled $y \in \{0, 1\}$ for the presence of slow activity [13]. We create a training set T derived from this set in Sequence generation. The validation dataset V contains $|V| = 12345$ ten second sequences sampled from 34 patients with the same EEG channels and sampling rate [13]. Each sequence is labeled $y \in \{0, 1\}$. The validation set V has a class imbalance for label y, with $|V_{pos}| = 3,219$ and $|V_{neg}| = 9,126$.

Inputs / output format

Inputs Detection of the onset of slow activity requires detection within the a short time-span in order to be clinically useful. A sequence length of 10 seconds was chosen based on this requirement. It follows that the input sequence to the model contains $\textbf{len}\ seq_{input} = 10r = 2000$ timesteps. Each training example contains ten sequences of pairwise offsets. Considering both the sequence length and number of channels, the input to our model has the shape $(\textbf{len}\ seq_{input}, |F|) = (2000, 10)$.

Outputs Our model estimates $P(y|x)$, which is a scalar value ranging between 0 and 1. Hence, the output of our model has the shape $(1,)$

Preprocessing

Sequence generation In order to make the maximum utilization of the original training data, we first create a set S_{pos} of as many positive sequences with length $\textbf{len } seq_{input} = 2000$ as possible for an individual patient, starting with $t_f = t_{onset}$, and stopping after $t_i = t_{onset}$. For memory efficiency, a stride of 5 was used during the creation of each sequence in S_{pos}. We then create a disjoint set S_{neg} by randomly sampling at most $|S_{pos}|$ negative sequences with replacement from a uniform distribution containing every possible negative training example (sequences with $t_f < t_{onset}$) from the same patient. This process is repeated for each patient, and the final training set T contains the union of each S_{pos} and S_{neg} set.

Resampling Before training, 50% of generated sequences were randomly cropped relative to the first timestep, resulting in a new sequence seq'_{input} with the relationship $\textbf{len } seq'_{input} = u\textbf{len } seq_{input}$, where $u \in [0.9, 1.1]$ is sampled from a uniform distribution. seq'_{input} was then resampled to the original length $\textbf{len } seq_{input} = 2,000$. While this is a commonly used image augmentation technique for object detection [14, 15], it should also be beneficial here since we are interested in augmenting the temporal relationship between frequency and phase rather than the frequency and phase itself.

Network architecture

Other researchers have demonstrated success with residual neural network variants for detecting complicated patterns in signals [2]. Thus, we use a similar variation of *ResNet* as a starting point with pre-activation style blocks [5] as shown in Figure 1. Through trial and error, the first few convolution layers use a $D = 32$ dimensional kernel, before increasing to $2D$ and ending with $4D$. Increasing $D = 32$ by factors of 2 resulted in overfitting. Likewise, reducing $D = 32$ by factors of 2 resulted in underfitting. With $D = 32$, our model has $p = 165664$ trainable parameters.

Anti-aliased temporal downsampling We explored several different methods of temporal downsampling in our network architecture, as well as investigating recent advancements in reducing aliasing [6]. After deciding on other hyper parameters, we trained our network with an anti-aliased version of strided downsampling. We use a three point Gaussian low pass kernel with $\sigma \approx 0.79577$ during downsampling. We use the same σ for each of the three downsampling operations to encourage the network to learn a feature representation increasingly focused on lower frequencies. Each downsampling operation divides the temporal axis of the sequence by two.

Global temporal downsampling Recent papers in deep learning have increasingly relied on global pooling layers to reduce the number of trainable parameters and improve generalization for a variety of problems [11, 16, 8]. We considered several different global downsampling strategies including global max pooling (GMP), global average pooling (GAP) [8], and flattening. GAP was excluded because it may not be able to effectively handle sequences where only a small percentage contains the onset. Flattening significantly increases the number of trainable parameters, and may bias towards certain parts of the sequence in the training set. GMP provides

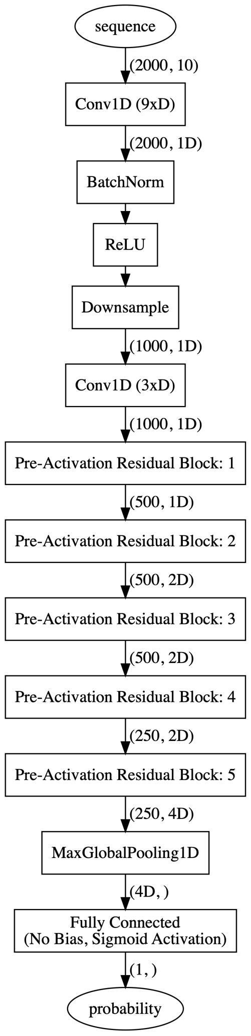

Figure 1 Our network has 12 convolutional layers, each of which is followed by batch normalization and a rectified linear unit. Residual connections are used to improve gradient propagation throughout the network.

the largest activation value from each channel regardless of where it occurred. With these considerations in mind, GMP was selected for global temporal downsampling on the top of the network.

Training

Online augmentation

During training, online augmentations were employed to help the network to learn how to handle differences in variance and bias from patient to patient. We employ global batch-wise z-scoring, when combined with a small stride size during sequence generation, smaller batch sizes, and sample-wise shuffling results in the network being forced to generalize to a considerable number of different scales and biases.

Global batch-wise z-scoring z-scoring was done along batch, temporal, and channel axes, normalizing the entire batch using a single mean and standard deviation. Let B_n be a mini batch of shape $(|B|, \textbf{len}\ seq_{input}, |F|) = (16, 2000, 10)$ for a batch size of 16. Each mini batch B_n is randomly sampled without replacement from a uniform distribution during the start of every training epoch. We calculate the mean μ_{batch} and standard deviation σ_{batch} by reducing all three axes to a single scalar value. We then apply standard z-scoring as follows $B'_{train} = \frac{B_{train} - \mu_{batch}}{\sigma_{batch}}$. B'_{train} is then used to calculate the loss during training. When validating our models performance, we instead z-score the validation set using the training set population mean and standard deviation.

Loss

Since our neural network is a binary classifier, we used a binary cross-entropy based cost function to train the network.

Kernel regularization In order to encourage the model to not overemphasize a small subset of learned features which may be biased towards the training set, we used L_2 kernel regularization. $\lambda = 0.01$ was chosen for the L_2 penalty for all convolution kernels using through trial and error [9].

Loss discount factor While GMP may help with cases where only a small part of the onset is present, some positive sequences generated using our methodology only contain a small number of positive time steps which may negatively impact convergence. If more data was available, we could simply omit ambiguous regions during training. Due to data paucity however, another solution is needed. We define a cost discounting function $\alpha(p)$ where p is defined as the number of positive time steps in a sequence divided by the total length of the sequence:

$$\alpha(p = \frac{n_{pos}}{\textbf{len}\ seq_{input}}) = \begin{cases} 0.95 & p = 0 \\ 10p & 0 < p \leq 0.1 \\ 1 & 0.1 < p \end{cases}$$

This effectively discounts loss during the first second after the onset, starting from complete discount at $t_{onset} = t_{final}$ and ending with no discount at $t_{onset} = t_{final} - r$, with our discount linearly decreasing as $t_{onset} \to t_{final} - r$. Since our classes are balanced, we chose to discount a proportional amount from all negative examples in order to avoid bias. Finally, we define our cost function as:

$$loss(y_{true}, y_{pred}) = \alpha \cdot bce(y_{true}, y_{pred}) + \lambda \sum_{i=1}^{p} \beta^2$$

Optimization

We optimized our network during training using mini-batch stochastic gradient descent (SGD).

Batch size We used a mini-batch size of 16 during each training step. While a much higher batch size could easily fit into memory during training, smaller batch sizes result in a wider range of scale and bias when utilizing batch-wise z-scoring. Smaller batch sizes have also been observed to have a regularizing effect on the model when training with SGD [10].

Training parameters We selected an initial learning rate of $\eta_i = 0.0001$, decaying by a factor of 2 every 15 epochs for a total of 75 epochs. Momentum was set to $\beta = 0.9$.

Experimental Setup

While developing our method, we observed a high variability of outcome with different random seeds. In order to test the reliability of our methods, we conducted ten runs using different random seeds with our method during training.

Method Variants

In addition to our full method, we applied the same experiment setup to different variants omitting batch-wise z-scoring, L_2 kernel regularization, and anti-aliased downsampling. For the z-scoring variant, we normalize each sequence with its own mean and standard deviation during training and validation. The L_2 variant simply omits the L_2 penalty. The method without anti-aliased down-sampling performs a strided down-sampling before the residual connection, and adds a max pooling layer on the residual in order to match the sequence lengths. Two additional variants use batch sizes of 32 and 64. Finally, we created a baseline variant without batch z-scoring, L_2 regularization, anti-aliased downsampling, and the discount factor. For this variant we selected to use a batch size of 64. All variants share the same ten random seeds used in the full method for comparison.

Results

Metrics

Due to class imbalance in the validation set, we use receiver operator characteristic area under curve (ROC-AUC) to evaluate the accuracy of our model. Despite the imbalance, are also interested in the trade off between sensitivity and specificity for each of our variants. To compute sensitivity and specificity, values of $y_{pred} > 0.5$ are considered true, and values of $y_{pred} \leq 0.5$ are considered false. The same threshold also applies for accuracy.

Average accuracy

Accuracy over ten training runs is shown in Table 1. Our full model had the highest average ROC-AUC and highest and most consistent accuracy out of each of our variants. In our variant which omitted batch-wise z-scoring, we observe a significant increase in metric variance as well as the lowest average sensitivity and ROC-AUC. We hypothesize there is not enough variance in scale and bias in the training set without this augmentation. The variant without L_2 regularization struggled with ROC-AUC and specificity, while having slightly higher average sensitivity than our full method. Even considering the fact that our model only has ≈ 165 K trainable parameters, without L_2 kernel regularization there is clear evidence that a small number of features are overemphasized. Our variant without anti aliasing has a higher sensitivity than our full method. However, this comes at a significant cost in specificity. We hypothesize that this is due to the model associating aliasing with the presence of the onset, and that anti-aliasing and/or removal of high frequency information is important for reducing the frequency of false positives. The variant without loss discounting was the closest to our best results, trading off more specificity than was gained in sensitivity. In both cases, increasing the batch size from 16 has a significant negative impact on ROC-AUC during validation. Our baseline model predictably had the worst results overall.

Table 1 Comparing our full method to methods which omit one technique: ten runs $\mu \pm \sigma$

Method varriant	ROC-AUC	Sensitivity	Specificity	Accuracy
baseline	0.600 ± 0.024	0.446 ± 0.081	0.689 ± 0.057	0.626 ± 0.026
batch size = 64	0.646 ± 0.027	0.468 ± 0.064	0.732 ± 0.042	0.664 ± 0.022
w/o L_2	0.651 ± 0.018	0.486 ± 0.064	0.747 ± 0.049	0.679 ± 0.028
w/o batch z-score	0.658 ± 0.034	0.379 ± 0.060	0.801 ± 0.055	0.691 ± 0.034
batch size = 32	0.689 ± 0.032	0.500 ± 0.069	0.759 ± 0.048	0.692 ± 0.027
w/o anti-aliasing	0.708 ± 0.017	0.527 ± 0.066	0.767 ± 0.051	0.708 ± 0.024
w/o discount	0.712 ± 0.026	0.462 ± 0.032	0.825 ± 0.037	0.731 ± 0.023
Full method	$\mathbf{0.725 \pm 0.025}$	$\mathbf{0.448 \pm 0.063}$	$\mathbf{0.850 \pm 0.032}$	$\mathbf{0.746 \pm 0.016}$

Maximum accuracy

Table 2 shows the best single validation ROC-AUC of each variant. We observe our full method has highest single epoch ROC-AUC of each variant. All of our variants appear to be heavily dependent on weight initialization and mini-batch batch selection during training, with many separate training runs needed to achieve highest generalization. We hypothesize that this is due to both the paucity of the data set and unstable gradients caused by lower batch sizes.

Table 2 Comparing our full method to methods which omit one technique: ten runs best validation

Method variant	ROC-AUC	Sensitivity	Specificity	Accuracy
baseline	0.639	0.613	0.562	0.575
w/o batch z-score	0.667	0.202	0.886	0.708
w/o L_2	0.680	0.539	0.794	0.727
batch size = 64	0.696	0.488	0.759	0.689
batch size = 32	0.725	0.636	0.667	0.659
w/o anti-aliasing	0.728	0.505	0.774	0.704
w/o discount	0.749	0.464	0.832	0.736
Full method	**0.768**	**0.486**	**0.884**	**0.781**

In addition to the ten runs for our full method, we conducted approximately twenty additional runs for our full method with new random seeds. In Table 3, We show the best overall model in terms of ROC-AUC. The model has much higher sensitivity without sacrificing a significant amount of specificity. We use this model for all following discussion and visualization of model behavior.

Table 3 Full method additional training runs: maximum ROC-AUC

Method variant	ROC-AUC	Sensitivity	Specificity	Accuracy
Full method	**0.772**	**0.606**	**0.828**	**0.770**

Explaining model predictions

Salience In order to help explain our models predictions, we computed the gradient of y with respect to input sequences from the test set and summed the absolute value of the gradient for each feature channel together:

$$salience(t) = \sum_{f=0}^{9} \left| \frac{\partial y}{\partial seq_{t,f}} \right|$$

For visualization purposes, we normalize salience with the timestep containing the maximum value: $salience_{vis}(t) = \frac{salience(t)}{salience(t_{max})}$. In each visualization we see only strong, sparse activation contributing to the models decision due to the GMP layer at the top of the network.

Example: true positive Arguably the strongest activation overall appears to happen when almost every channel simultaneously increases, which can happen several times around the onset. We visualize this in Figure 2, where observe strong activation on the rising edge of a global increase.

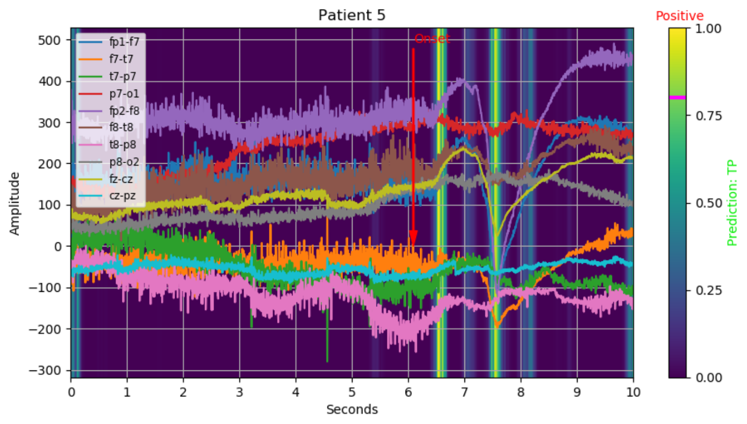

Figure 2 Salience: $y_{true} = 1, \lfloor y_{pred} \rceil = 1$ (True Positive)

Example: false negative Only some instances of the onset of slow activity exhibit strong cross channel correlation, as demonstrated in Figure 3. While most channels appear to move simultaneously, there is less positive correlation as well as some negative correlation between channels. In this particular example, there appears to be a wide spread of channel bias and low dynamic range. We hypothesis that z-scoring using the population mean and standard deviation may not be optimal for all examples, and that an adaptive strategy could improve validation performance.

Example: false positive Figure 4 demonstrates that not all instances of cross channel correlation are useful for predicting the onset by themselves. We hypothesize that a model may need to take into account the temporal nature of the problem in order to avoid these types of false positives.

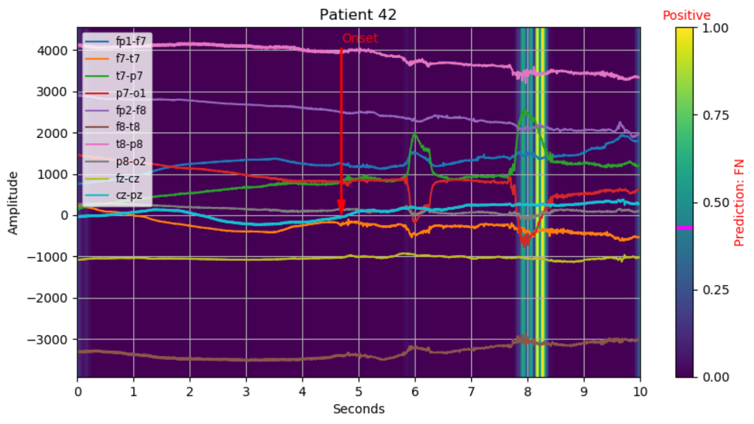

Figure 3 Salience: $y_{true} = 1, \lfloor y_{pred} \rceil = 0$ (False Negative)

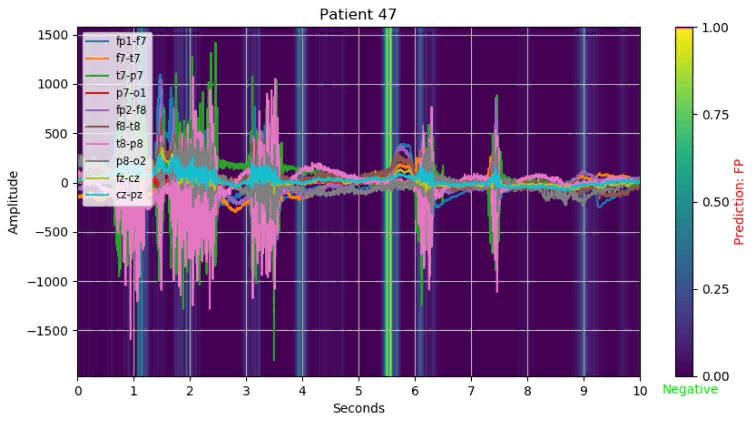

Figure 4 Salience: $y_{true} = 0, \lfloor y_{pred} \rceil = 1$ (False Positive)

Implementation

We implemented our model using Python 3.7 and tensorflow.keras [17]. Our full source code is available on Github: https://github.com/csvance/deep-onset-detection

Conclusion

While our naive baseline model had relatively poor accuracy, we demonstrated the impact of many different regularization techniques. It follows that deep learning can be an effective tool for signal detection problems with a small amount of available training data. By conducting our experiment over many different training runs, we show the statistical significance of our results. Finally, we demonstrated that while our model may be a black box, we can make the results easier to interpret with salience and effective visualization.

Future work

We recognize that the loss discount factor could be made into a continuous function across the entire sequence. Currently, examples with a negative label could contain the start of the onset due to the the labeling task being particularly challenging, but are weighted as heavily as non ambiguous examples. In addition, we observed examples of false positives which would be relatively easy for a human to classify correctly due to drastic changes in overall behavior patterns. An improved model would be able to recognize these changes over time in addition to identifying channel cross correlation. We also recognize that generalisation performance may be improved by adding a batch normalization and final rectified linear unit before the GMP layer at the top of the network.

Competing interests
The authors declare that they have no competing interests.

Author details
[1]University of Houston, Houston, USA. [2]School of Biomedical Informatics, UT Health, 7000 Fannin St Suite 600, Houston, Texas, USA. [3]Department of Neurology, McGovern Medical School, UT Health, 6430 Fannin St, Houston, Texas, USA.

References
1. Krizhevsky, A., Sutskever, I., Hinton, G.E.: Imagenet classification with deep convolutional neural networks. In: Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1. NIPS'12, pp. 1097–1105. Curran Associates Inc., Red Hook, NY, USA (2012)
2. Rajpurkar, P., Hannun, A.Y., Haghpanahi, M., Bourn, C., Ng, A.Y.: Cardiologist-level arrhythmia detection with convolutional neural networks. CoRR **abs/1707.01836** (2017). 1707.01836
3. Ching, T., Himmelstein, D.S., Beaulieu-Jones, B.K., Kalinin, A.A., Do, B.T., Way, G.P., Ferrero, E., Agapow, P.-M., Zietz, M., Hoffman, M.M., Xie, W., Rosen, G.L., Lengerich, B.J., Israeli, J., Lanchantin, J., Woloszynek, S., Carpenter, A.E., Shrikumar, A., Xu, J., Cofer, E.M., Lavender, C.A., Turaga, S.C., Alexandari, A.M., Lu, Z., Harris, D.J., DeCaprio, D., Qi, Y., Kundaje, A., Peng, Y., Wiley, L.K., Segler, M.H.S., Boca, S.M., Swamidass, S.J., Huang, A., Gitter, A., Greene, C.S.: Opportunities and obstacles for deep learning in biology and medicine. Journal of The Royal Society Interface **15**(141), 20170387 (2018). doi:10.1098/rsif.2017.0387. https://royalsocietypublishing.org/doi/pdf/10.1098/rsif.2017.0387
4. Marcus, G.: Deep learning: A critical appraisal. CoRR **abs/1801.00631** (2018). 1801.00631
5. He, K., Zhang, X., Ren, S., Sun, J.: Identity mappings in deep residual networks. CoRR **abs/1603.05027** (2016). 1603.05027
6. Zhang, R.: Making convolutional networks shift-invariant again. CoRR **abs/1904.11486** (2019). 1904.11486
7. Ioffe, S., Szegedy, C.: Batch normalization: Accelerating deep network training by reducing internal covariate shift. CoRR **abs/1502.03167** (2015). 1502.03167
8. Lin, M., Chen, Q., Yan, S.: Network in network. CoRR **abs/1312.4400** (2013)
9. Schmidhuber, J.: Deep learning in neural networks: An overview. CoRR **abs/1404.7828** (2014). 1404.7828
10. Breuel, T.M.: The effects of hyperparameters on SGD training of neural networks. CoRR **abs/1508.02788** (2015). 1508.02788
11. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. CoRR **abs/1512.03385** (2015). 1512.03385
12. LeCun, Y., Haffner, P., Bottou, L., Bengio, Y.: Object recognition with gradient-based learning. In: Shape, Contour and Grouping in Computer Vision, p. 319. Springer, Berlin, Heidelberg (1999)
13. Jiang, X., Kim, Y.: SBMI Healthcare Machine Learning Hackathon. School of Biomedical Informatics (2019). https://sbmi.uth.edu/hackathon/archive/sept19.htm
14. Zhao, Z., Zheng, P., Xu, S., Wu, X.: Object detection with deep learning: A review. CoRR **abs/1807.05511** (2018). 1807.05511
15. Shorten, C., Khoshgoftaar, T.M.: A survey on image data augmentation for deep learning. Journal of Big Data **6**(1), 60 (2019). doi:10.1186/s40537-019-0197-0
16. Lathuilière, S., Mesejo, P., Alameda-Pineda, X., Horaud, R.: A comprehensive analysis of deep regression. CoRR **abs/1803.08450** (2018). 1803.08450
17. Chollet, F., et al.: Keras. https://keras.io (2015)