BME517 Final Project Report

Morgan Sun

1. Introduction

When dealing with applications for neural data, such as brain computer interfacing, neuroprosthetics, and other implantable neural devices, it is typically impractical to directly decode raw electrical signals from the electrode, as the data is too noisy and high-dimensional to be of use. Instead, it is often desirable to extract more interpretable features from the raw data for downstream decoding. One commonly used feature is constructed by counting the number of spikes from each single unit source. However, since each electrode may record spikes from multiple single unit sources, it is necessary to sort detected spikes.

1D convolutional neural networks (CNNs) have been shown to provide promising spike sorting accuracy. For example, Li et. al. [1] propose a CNN architecture which achieves over 99% accuracy when applied to the synthetic wave_clus dataset, and over 90% accuracy when applied to recorded neural data. However, their proposed architecture is sizable, placing relatively high demands on hardware speed, memory, and power. This reduces practicality for real-time implantable sorting.

Reducing the size of the proposed architecture could potentially decrease these hardware demands and make CNNs more practical for implantable applications; however, it is unclear how sorting accuracy would be affected by such reductions. The goal of this project is therefore twofold: first to validate and replicate their experiments on the wave_clus dataset, and second to investigate and compare various strategies for reducing the computational footprint of the architecture.

2. Methods

Section 2.1 will present the data used for this project, as well as the ways in which it was preprocessed and prepared. Section 2.2 will discuss the implementation of a simple baseline model for the purpose of benchmarking and comparison. Section 2.3 will present the methods for validation of the architecture proposed by Li et. al., while Section 2.4 will explain the model size reduction strategies tested.

2.1. Data and Preprocessing

The wave_clus collection of synthetic datasets is relatively popular for benchmarking spike sorting algorithms [2]. The datasets are synthetically generated using 594 spikes recorded from single sources, which are used as templates for both the spikes and background noise in creating the dataset. The collection can be divided into four difficulty levels (“Easy1”, “Easy2”, “Difficult1”, and “Difficult2”). Each difficulty level contains multiple datasets with various signal-to-noise ratios; in total the collection includes 20 datasets. Figs. 1-4 show some examples of spikes from the “Easy1” set.

Since the datasets are synthetic, they come with ground-truth labels for spike times and categories. Following the methodology of Li et. al., time labels were used to isolate 64-sample windows containing individual time-aligned spikes. These 64 samples serve as the set of input features to all the algorithms and models tested in this project. Each dataset within the collection contained slightly over 3000 individual spikes divided evenly across three clusters.

Each dataset was split in half, with the first half used as training data and the second half as test data. It is worth noting that, due to the synthetic nature of the dataset, the spikes from either half can be considered to have

been sampled from the same distribution. Nonetheless, for each dataset, both the training and testing splits were normalized by subtracting the training mean and dividing by the training standard deviation.

The 20 datasets within wave_clus were not mixed into one large dataset; instead, they were used independently. In other words, for each algorithm/architecture tested, separate copies were trained and evaluated on each of the 20 datasets. Some experiments required model hyperparameter tuning; for these experiments, hyperparameters were tuned by optimizing average performance on the Easy1-0.15 and Difficult1-0.15 test sets, essentially treating them as validation data, since this was the approach taken by Li et. al.

2.2. PCA + GMM Clustering

To serve as a benchmark for comparing the CNN-based models, a relatively simple clustering algorithm based on more conventional techniques was implemented. This clustering algorithm first uses principal component analysis (PCA) to reduce spike dimensionality from 64 to n_{pca}, where n_{pca} is a hyperparameter optimized using grid search. A Gaussian mixture model (GMM) is then used to perform clustering. For both PCA and GMM, the implementations from the scikit-learn package were used. Performance was measured via classification accuracy. As GMM does not necessarily generate the same cluster labels as those used in ground truth labels, accuracy was evaluated six times using all possible remappings of cluster labels, and the best accuracy out of six was used.

2.3. Baseline CNN Architecture and Training

Since the code for Li et. al.'s experiments is not publicly available, it was necessary to replicate their model from scratch in TensorFlow Keras. However, although their paper describes their model's layers and activation functions, it does not elaborate on several details, including the optimizer and learning rate, initialization methods for layer weights, and batch size. Additionally, the wording used when describing their usage of BatchNorm and Dropout is ambiguous, and it is unclear exactly where in the model these regularization layers were added.

For the sake of replicating their model as best as possible, these ambiguities were treated as hyperparameters to optimize. For the sake of limiting the scope of this search, the optimizer was assumed to be Adam, weight initialization was assumed to be Glorot uniform, and batch size was assumed to be 256. The remaining hyperparameters were optimized using a grid search; the space that was searched over is described by Table 1, with the best hyperparameters highlighted.

The final CNN architecture after optimization is described in Table 2. Models were trained to optimize a categorical crossentropy loss function, and additionally evaluated on categorical accuracy. Training continued until terminated by an early stopping callback, which typically occurred within around 10 epochs.

2.4. CNN Size Reduction Experiments

Four strategies for reducing the memory and computational cost of the baseline CNN architecture were tested. The former two strategies involve decreasing the number of filters and neurons used in the convolutional and dense layers, respectively. The latter two strategies, which are mutually exclusive of each other, involve increasing the intensity of max-pooling, either by increasing pooling window size or adding additional pooling layers.

Each strategy is tested at a stepped series of progressively increasing levels of aggression; Tables 3-6 describe these levels in more detail.

All strategies were tested both in isolation (e.g., only decreasing convolutional filter counts), as well as in combination with each other (e.g., decreasing both convolutional filter counts and dense layer neuron counts), yielding 6 additional hybrid strategies for a total of 11 strategies.

For each level of each strategy, training was repeated 5 times per dataset for each level of each strategy, and the mean test accuracy across these $20 \times 5 = 100$ training runs was used as an accuracy metric. Additionally, the memory cost of the model was measured via the number of trainable parameters, and the runtime cost of the model was measured in the number of floating-point multiplications necessary for the forward pass.

3. Results and Discussion

Section 3.1 will compare the results of the baseline models introduced in Sections 2.2 and 2.3. Section 3.2 will discuss the results of the CNN size reduction experiments.

3.1. Baseline Performances

As shown in Figure 5, the optimal number of dimensions to use in the PCA+GMM clustering model was found to be 14. The performances of the PCA+GMM model, the baseline CNN model as implemented in this project, and the accuracies reported by Li et. al. on the individual wave_clus datasets are reported in Table 7.

Overall, the performance of the relatively simple PCA+GMM model remains high for the easiest datasets in the wave_clus collection, but this model does not remain robust in the face of increasing noise and difficulty level.

The from-scratch implementation of the baseline CNN model architecture achieves very high accuracy for all datasets. Performance still falls slightly short of the metrics reported by Li et. al., which may be due to differences in hyperparameters such as initialization method, optimizer algorithm, and batch size; nonetheless, since the objective of this project is to investigate how model performance is impacted by architectural simplifications in relative terms, this slight difference in performance doesn't seem to invalidate the key conclusions of this project.

3.2. CNN Size Reduction Results

Figures 6 and 7 show the response of the CNN model to various combinations of model size reduction strategies. More specifically, Figure 6 plots the number of model parameters against error rate, while Figure 7 plots the number of floating-point multiplications against error rate. As each line in the plot moves from right to left, the number of parameters and multiplications required for the model decreases. An ideal method of reducing model size would be able to drastically reduce the number of parameters and multiplications while not incurring substantial decreases in error rate, which would result in a line running horizontally from right to left across the bottom of the plot. However, all actual methods of reducing model size must eventually begin inducing reductions in model accuracy. Therefore, the lines shown in Figures 6 and 7 begin at the lower right-hand corner and move towards the upper left-hand corner as models shrink and performances degrade.

The figures show that there appears to be a very high degree of correlation between the number of parameters and the number of multiplications. This is unsurprising, especially since most parameters and multiplications in the baseline CNN model are concentrated in the dense layers, where there is a 1-to-1 relationship between parameter and multiplication count.

The figures also show that increasing pooling is the least effective single strategy for shrinking model size, as both methods of increasing pooling are only able to reduce the model's size by around half an order of magnitude. On the other hand, reducing the number of kernels and neurons in convolutional and dense layers can dramatically reduce the memory and computational cost of running the CNN model. When using both strategies simultaneously, it is possible to reduce the model's footprint by around two orders of magnitude while suffering no adverse effects on accuracy. In fact, applying these strategies at a less aggressive level appears to have a regularizing effect, slightly increasing accuracy.

4. Conclusion

In this project, experiments were done to validate the CNN model architecture proposed by Li et. al. for neural spike sorting. By re-implementing their methodology from scratch, accuracy metrics very similar to their reported results were achieved. Furthermore, it was found that their model can be made more efficient, in terms of both memory and computational cost, by around two orders of magnitude without suffering any noticeable degradation in accuracy. Such a reduction in the cost of running a model greatly impacts the practicality of deploying such a model in the context of a medical implant, where memory, compute, and power are greatly constrained.

However, this project only replicates the experiments performed by Li et. al. on the synthetic wave_clus dataset. Li et. al. additionally evaluate their model on a dataset of real biologically sourced data, but this project did not replicate these experiments and therefore it is unclear whether the reduced-size CNN models discussed in this project would perform in a robust manner when faced with non-synthetic data. The work performed in this project could therefore be extended by trying to apply the same methodology to more realistic biologically sourced data as opposed to synthetic benchmark datasets.

Additionally, this project only explores relatively straightforward architectural tweaks to the model architecture proposed by Li et. al., which is itself a relatively "vanilla" CNN model design. Many variations on the CNN idea and neural networks in general exist which might be promising alternatives for lightweight neural-network-based spike sorting. For example, binarized [3] and spiking [4] neural networks seek to reduce cost using drastic changes to the mathematics or hardware underlying the network, while other techniques such as global-average-pooling [5] are promising methods to eliminate the costly dense layers used in conventional CNN designs while also improving model generalizability. It would be worthwhile to explore these additional methods for lightweight spike sorting.

Appendix A: Lab Code

The code for this project is not directly pasted here, but instead has been made available on Github at the following link. The key prerequisites are numpy, pandas, matplotlib, scipy, scikit-learn, and tensorflow.

https://github.com/morgannewellsun/bme_final_project

Appendix B: References

[1] Li, Z. *et al.* (2020) "An accurate and robust method for spike sorting based on Convolutional Neural Networks," *Brain Sciences*, 10(11), p. 835. Available at: https://doi.org/10.3390/brainsci10110835.

[2] Quiroga, R.Q., Nadasdy, Z. and Ben-Shaul, Y. (2004) "Unsupervised spike detection and sorting with wavelets and superparamagnetic clustering," Neural Computation, 16(8), pp. 1661–1687. Available at: https://doi.org/10.1162/089976604774201631.

[3] Courbariaux, M. et al. (2016) "Binarized Neural Networks: Training Deep Neural Networks with Weights and Activations Constrained to +1 or -1."

[4] Tavanaei, A. et al. (2019) "Deep learning in spiking neural networks," Neural Networks, 111, pp. 47–63. Available at: https://doi.org/10.1016/j.neunet.2018.12.002.

[5] M. Lin, Q. Chen, and S. Yan. Network in network. International Conference on Learning Representations, 2014. 1, 2, 4

Appendix C: Tables

Table 1: Hyperparameter Grid Search Space

Hyperparameter	Options				
BatchNorm usage	Before all Conv layers	Before first Dense layer	Before all Dense layers	Before all Conv and Dense layers	
Dropout usage	Before all Conv layers	Before first Dense layer	Before all Dense layers	Before all Conv and Dense layers	
Dropout rate	0.1	0.2	0.3	0.4	0.5
Learning rate	0.0001	0.001	0.01		

Table 2: Final Baseline CNN architecture

Layer	Activation	Window	Stride	Input	Output
Input	-	-		64	
Reshape	-	-		64	(64, 1)
Conv #1	ReLU	3	1	(64, 1)	(64, 32)
Conv #2	ReLU	3	1	(64, 32)	(64, 64)
Pool #3	-	2	2	(64, 64)	(32, 64)
Conv #3	ReLU	3	1	(32, 64)	(32, 128)
Pool #4	-	2	2	(32, 128)	(16, 128)
Conv #4	ReLU	3	1	(16, 128)	(16, 128)
Flatten	-	-		(16, 128)	2048
Dropout (0.5)	-	-		2048	
BatchNorm	-	-		2048	
Dense #1	ReLU	-		2048	300
Dense #2	ReLU	-		300	100
Output	SoftMax	-		100	3

Table 3: Convolutional Layer Kernel Reduction Levels

	Number of Kernels					
Layer	**Baseline**	**Level 1**	**Level 2**	**Level 3**	**Level 4**	**Level 5**
Conv #1	32	16	8	4	2	1
Conv #2	64	32	16	8	4	2
Conv #3	128	64	32	16	8	4
Conv #4	128	64	32	16	8	4

Table 4: Dense Layer Neuron Reduction Levels

	Number of Neurons					
Layer	**Baseline**	**Level 1**	**Level 2**	**Level 3**	**Level 4**	**Level 5**
Dense #1	300	150	75	36	18	9
Dense #2	100	50	25	12	6	3

Table 5: Pooling Window Size Increase Levels

	Window Size						
Layer	**Baseline**	**Level 1**	**Level 2**	**Level 3**	**Level 4**	**Level 5**	**Level 6**
Pool #3	2	2	4	4	4	8	8
Pool #4	2	4	2	4	8	4	8

Table 6: Pooling Layer Frequency Increase Levels

Baseline	**Level 1**	**Level 2**	**Level 3**	**Level 4**	**Level 5**	**Level 6**	**Level 7**
Input	Input	Input	Input	Input	Input	Input	Input
Reshape	Reshape	Reshape	Reshape	Reshape	Reshape	Reshape	Reshape
				Pool #1	Pool #1	Pool #1	Pool #1
Conv #1	Conv #1	Conv #1	Conv #1	Conv #1	Conv #1	Conv #1	Conv #1
		Pool #2	Pool #2			Pool #2	Pool #2
Conv #2	Conv #2	Conv #2	Conv #2	Conv #2	Conv #2	Conv #2	Conv #2
Pool #3	Pool #3	Pool #3	Pool #3	Pool #3	Pool #3	Pool #3	Pool #3
Conv #3	Conv #3	Conv #3	Conv #3	Conv #3	Conv #3	Conv #3	Conv #3
Pool #4	Pool #4	Pool #4	Pool #4	Pool #4	Pool #4	Pool #4	Pool #4
Conv #4	Conv #4	Conv #4	Conv #4	Conv #4	Conv #4	Conv #4	Conv #4
	Pool #5		Pool #5		Pool #5		Pool #5
Flatten	Flatten	Flatten	Flatten	Flatten	Flatten	Flatten	Flatten
Dropout	Dropout	Dropout	Dropout	Dropout	Dropout	Dropout	Dropout
B.N.	B.N.	B.N.	B.N.	B.N.	B.N.	B.N.	B.N.
Dense #1	Dense #1	Dense #1	Dense #1	Dense #1	Dense #1	Dense #1	Dense #1
Dense #2	Dense #2	Dense #2	Dense #2	Dense #2	Dense #2	Dense #2	Dense #2
Output	Output	Output	Output	Output	Output	Output	Output

Table 7: Performance of Baseline Models (sub-95% accuracies in red)

Difficulty	Noise	PCA+GMM	Baseline CNN	Li et. al.
Easy1	0.05	0.976096	0.99317	0.9977
Easy1	0.1	0.969336	0.990346	0.9994
Easy1	0.15	0.994825	0.99425	0.9977
Easy1	0.2	0.993667	0.982729	0.9988
Easy1	0.25	0.992723	0.995755	0.997
Easy1	0.3	0.647871	0.991369	0.9971
Easy1	0.35	0.628749	0.992077	0.996
Easy1	0.4	0.676905	0.992912	0.9982
Easy2	0.05	0.634018	0.990616	0.9988
Easy2	0.1	0.645455	0.995455	0.9983
Easy2	0.15	0.652403	0.996483	0.9977
Easy2	0.2	0.643789	0.990357	0.9966
Difficult1	0.05	0.91844	0.945035	0.9905
Difficult1	0.1	0.346288	0.889791	0.9983
Difficult1	0.15	0.628456	0.985599	0.9942
Difficult1	0.2	0.361453	0.961336	0.9883
Difficult2	0.05	0.924495	0.996433	0.9982
Difficult2	0.1	0.967649	0.994223	0.9994
Difficult2	0.15	0.640116	0.993023	0.9971
Difficult2	0.2	0.645106	0.994276	0.9983

Appendix D: Figures

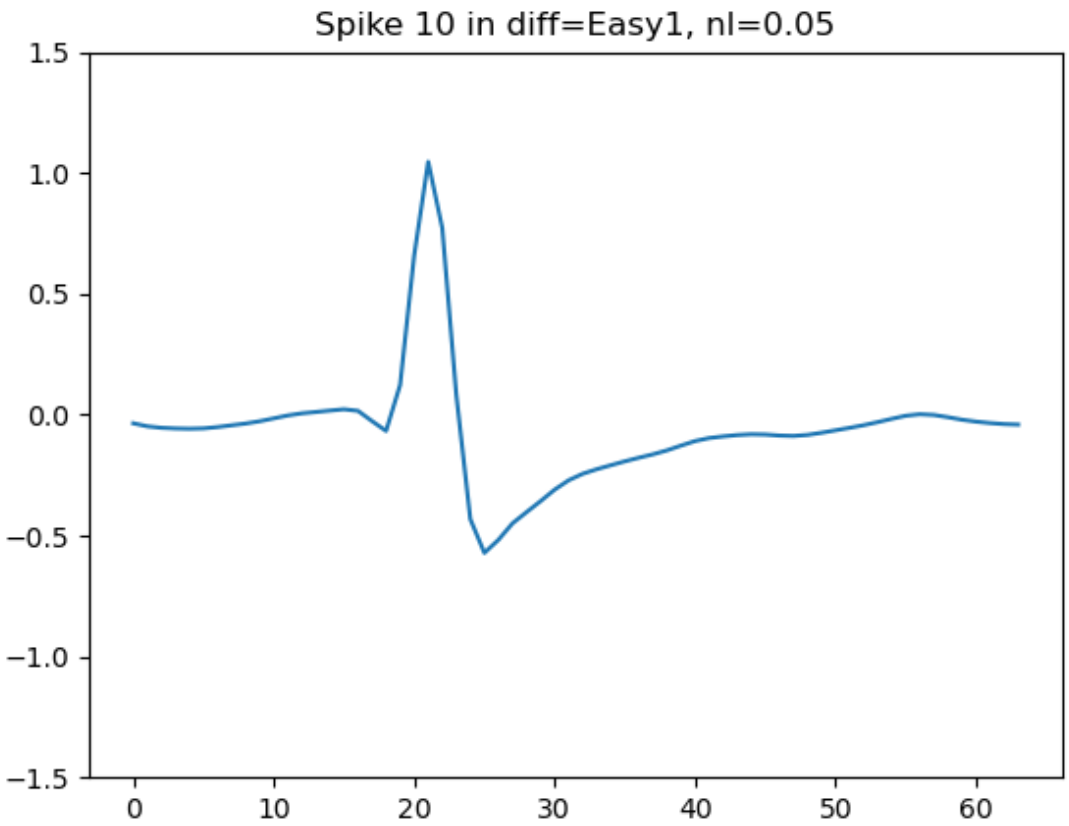

Fig 1: Example of a spike from the "Easy1" dataset with noise level 0.05

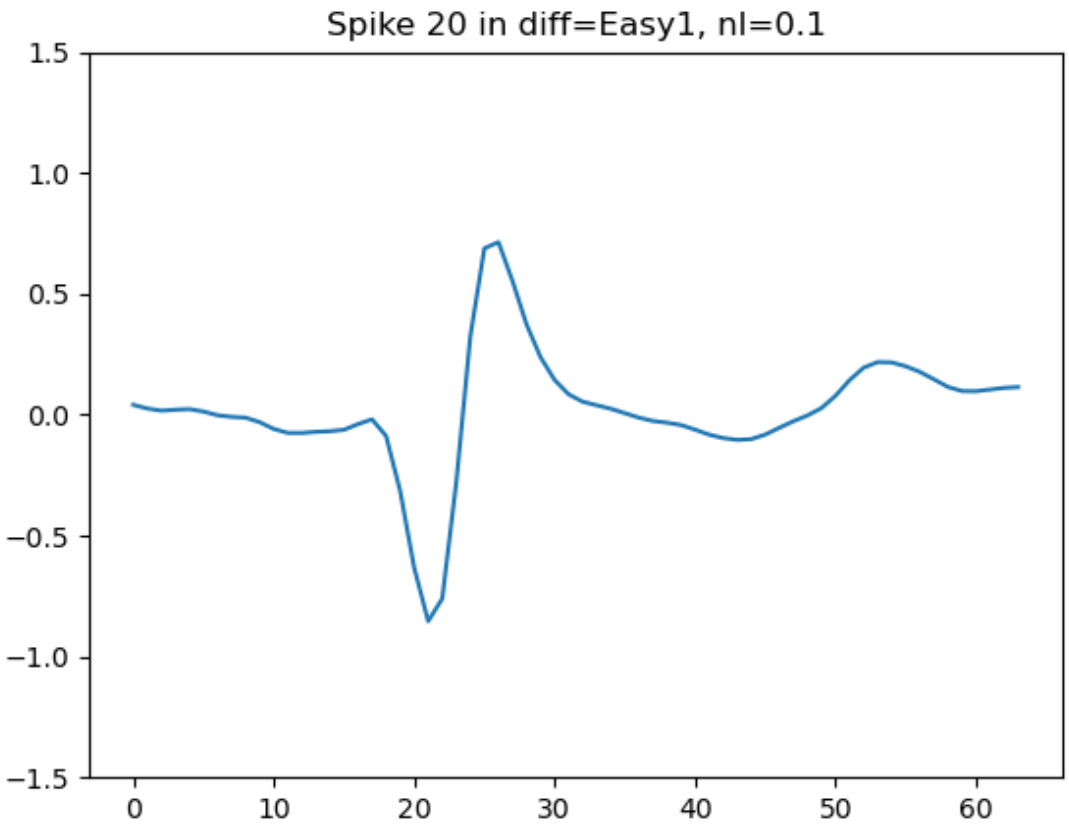

Fig 2: Example of a spike from the "Easy1" dataset with noise level 0.10

Appendix D: Figures

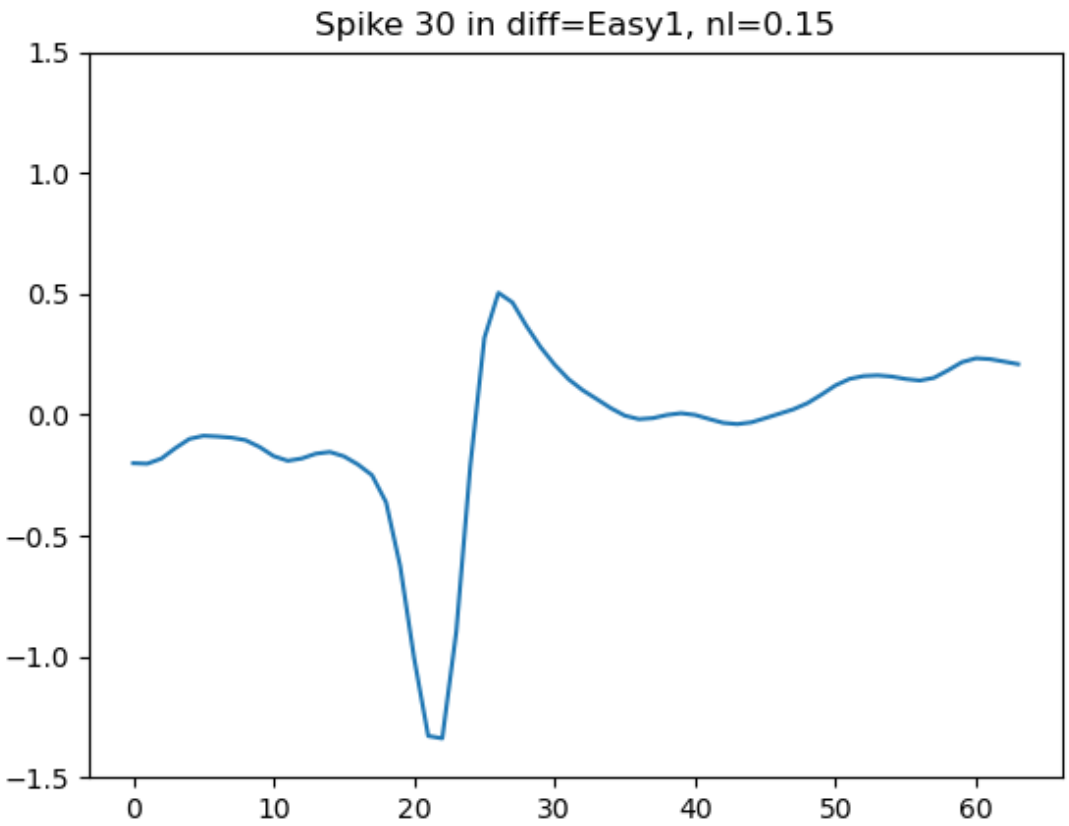

Fig 3: Example of a spike from the "Easy1" dataset with noise level 0.15

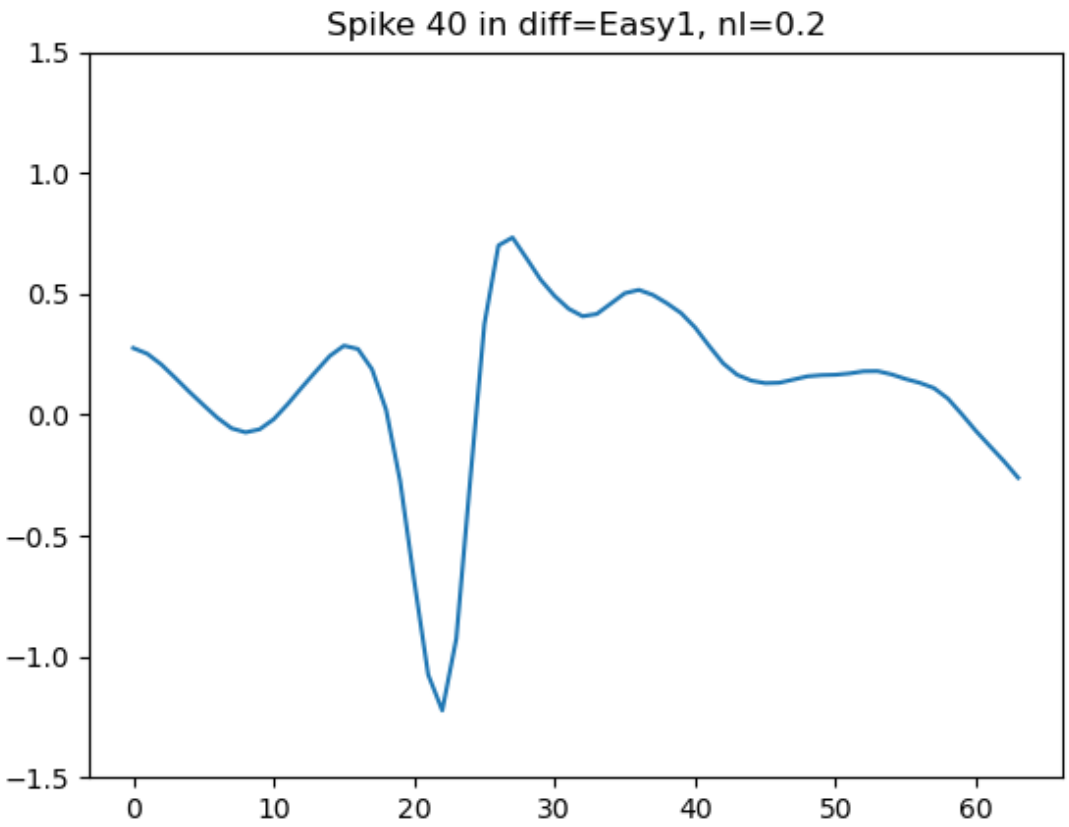

Fig 4: Example of a spike from the "Easy1" dataset with noise level 0.20

Appendix D: Figures

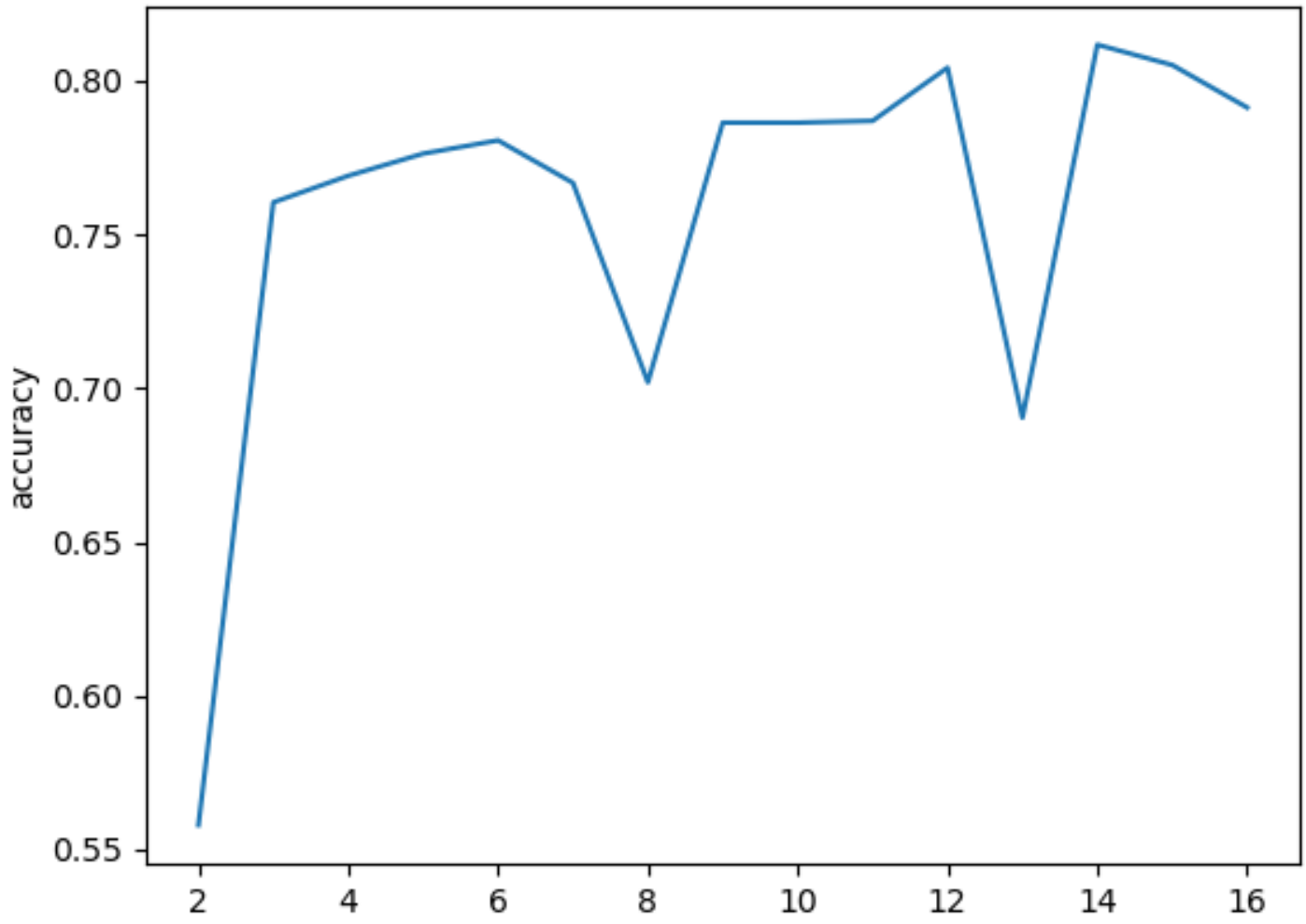

Fig 5: Grid search over number of dimensions for PCA+GMM clustering model

Appendix D: Figures

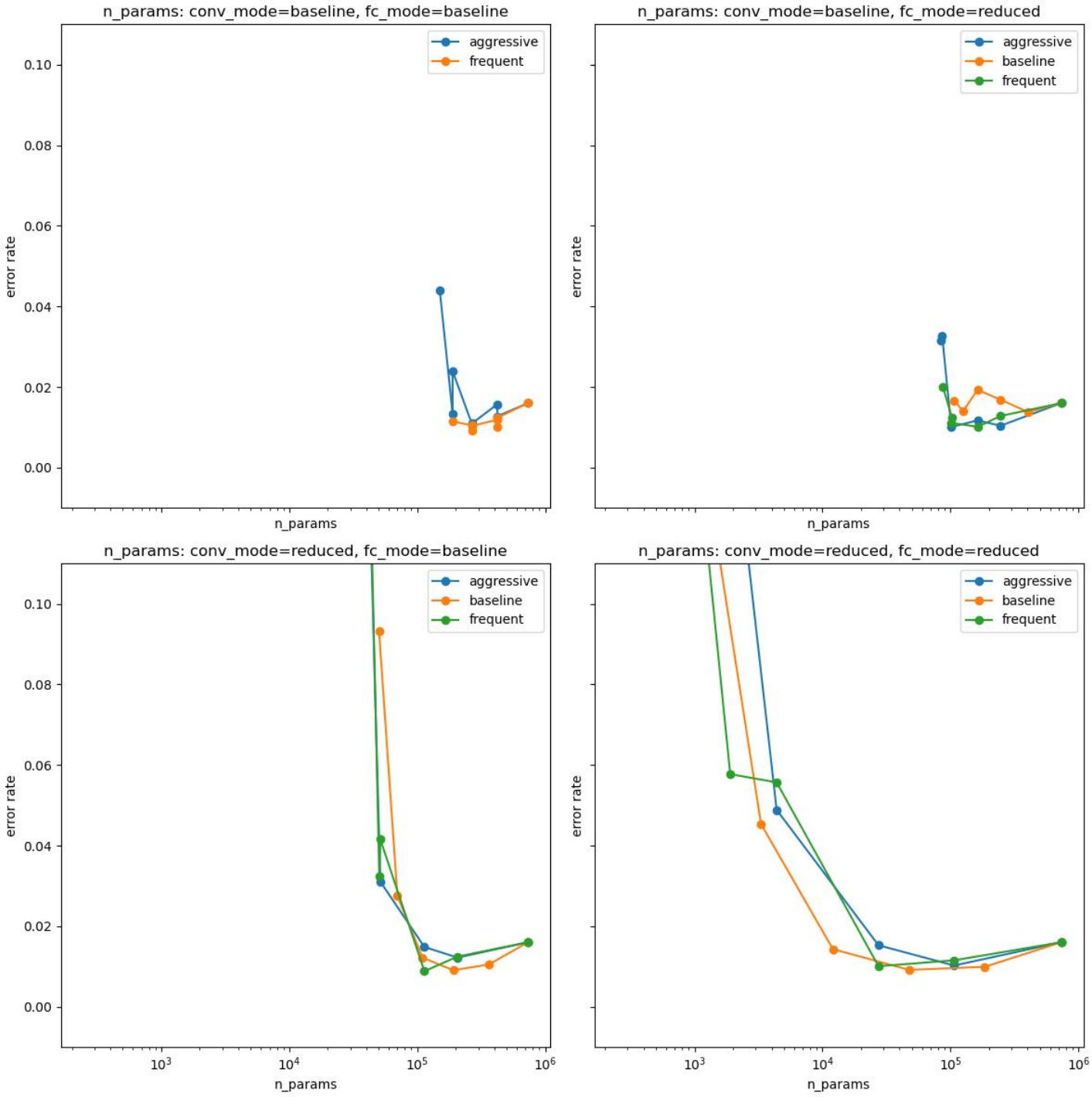

Fig 6: CNN parameter count vs error rate

Appendix D: Figures

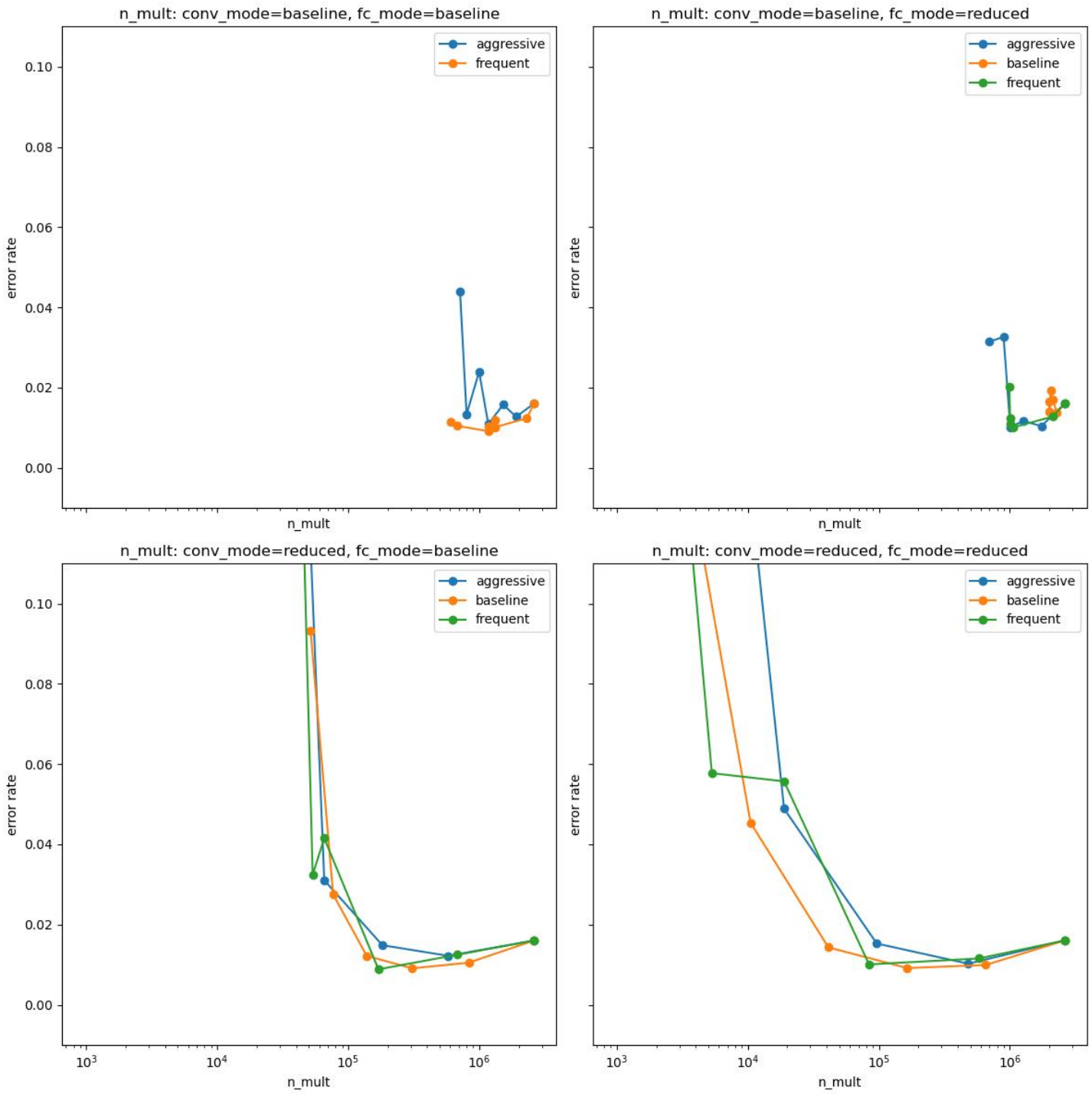

Fig 7: CNN forward pass multiplication count vs error rate