How to design a good classifier for EEG temporal data using CNNs and RNNs

Xiaoran Zhang
UCLA ECE
xiaoran108@ucla.edu

Hexiang Dong
UCLA ECE
bobdong233@ucla.edu

Abstract

*In this paper, we implemented nine different EEG classification architectures and compared the results. We also evaluated the effectiveness of different data augmentation and normalization schemes. Our best model after tuning achieves a test accuracy of **72.01%**.*

1. Introduction

Deep learning-based approaches have enabled the possibility of extracting information from electroencephalography (EEG) recordings of brain activity, which could be applied to many clinical applications [2]. There are especially many successful cases in EEG decoding using deep learning-based architectures. In this paper, we implemented nine different deep learning-based classification networks and various normalization and augmentation approaches to improve the classification accuracy. We also discuss several observations in the experiments and explore the possibility of future improvements.

We firstly followed the architectures proposed by Schirrmeister *et al*. [2] by implementing Shallow-CNN and Deep-CNN. We then test several RNN architectures including LSTM and GRU. Inspired by Wang *et al*. [3] that a general CNN-RNN architecture will produce a higher classification accuracy, we implemented the Shallow-CNN-LSTM and Shallow-CNN-GRU for comparison. We don't continue the fashion for Deep-CNN because of the observation that Deep-CNN has already had a complex architecture. Adding more trainable parameters into the architecture will result in an overfitting. Apart from reproducing architectures in the literature, we also create several novel architecture candidates. U-net has achieved a tremendous success in various biomedical applications after proposed by Ronneberger *et al*. [1] in 2015. One of the advantages of U-net's success is that the concatenation of detailed regional features and global features, which not only create a gradient highway but also incorporates more information for the network. We implemented a shallow U-shaped CNN (CNN-U) and several RNN variations including CNN-U-LSTM and CNN-U-GRU following the idea in [3] for comparison.

In order to design and select the best model to achieve a high test accuracy, we design the following pipeline. Firstly, we analyze various data normalization and augmentation schemes on the Shallow-CNN model. Then, we utilize the selected preprocessing approach and train different architectures with 1000 epochs. We compute the validation score to select several best model candidates. Finally, we tune the hyperparameters on the selected candidates and select the model with validation accuracy higher than 70% and report its test score as the best test accuracy. We also compared the results of optimizing different architectures on a single subject versus all subjects.

2. Methods

2.1. Data normalization

2.1.1 Min-max feature scaling normalization

The min-max feature scaling is a commonly used normalization scheme to scale the input data in $[0, 1]$. Suppose X is the input data and $X \in \mathbb{R}^{22\times1000}$, the normalized output X_n can be expressed as:

$$X_n = \frac{X - \min}{\max - \min}. \tag{1}$$

2.1.2 Standard score normalization

The standard score normalization, as known as z-normalization, scales the input data with zero-mean and unit standard deviation. The normalized output X_n could be expressed as

$$X_n = \frac{X - \mu}{\sigma}. \tag{2}$$

2.2. Data augmentation

2.2.1 Noise injection

We scale the training data to a factor of 0.5 and inject a Gaussian distributed noise $\mathcal{N}(0,\ 1)$ following to a randomly selected indices with probability 0.5. We then upsamples the data to original.

2.2.2 Random crop

Following the idea that having more training data generally improves the network performance [4], we generate more training samples using a sliding window random crop on the timestep axis. We set the crop size to 800 and generate 10 crops using a single training sample. We then upsamples the data to 1000 in timestep axis to match the dimension.

2.3. Model architecture

The details of our implemented architectures are shown in **Architecture 1-9**.

Architecture 1: Shallow-CNN

$inputs$ = **Input**(shape = (22, 1000))
r_1 = **Reshape**((1, 22, 1000))($inputs$)
c_1 = **Conv2D**((40, kernel = (1, 25), stride = 1))(r_1)
r_2 = **Reshape**((1, 880, 976))(c_1)
c_2 = **Conv2D**((40, kernel = (880, 1), stride = 1))(c_1)
sq_1 = **Activation**(Ksquare)(c_2)
r_3 = **Reshape**((1, 40, 976))(sq_1)
ap_1 = **AveragePool2D**((1, 75), stride = (1, 15))(r_3)
log_1 = **Activation**(Klog)(ap_1)
f_1 = **Flatten**()(log_1)
$outputs$ = **Dense**(4, activation = softmax)(f_1)

Architecture 2: Shallow-CNN-LSTM

$inputs$ = **Input**(shape = (22, 1000))
r_1 = **Reshape**((1, 22, 1000))($inputs$)
c_1 = **Conv2D**((40, kernel = (1, 25), stride = 1))(r_1)
r_2 = **Reshape**((1, 880, 976))(c_1)
c_2 = **Conv2D**((40, kernel = (880, 1), stride = 1))(c_1)
sq_1 = **Activation**(Ksquare)(c_2)
r_3 = **Reshape**((1, 40, 976))(sq_1)
ap_1 = **AveragePool2D**((1, 75), stride = (1, 15))(r_3)
log_1 = **Activation**(Klog)(ap_1)
r_4 = **Reshape**(40, 61)(log_1)
l_1 = **LSTM**(units = 128)(r_4)
$outputs$ = **Dense**(4, activation = softmax)(l_1)

3. Implementation Details

The models are trained and tested using NVIDIA RTX 2080ti GPU with 11GB memory. The environment is built on CUDA 10.0 with cuDNN v7.6.5 and keras 2.2.4 with tensorflow-gpu 1.14. The models are trained using Adam optimizer (learning rate = 1e-4, decay = 1e-6) with sparse categorical crossentropy loss function in Keras. The validation split is 0.8. The model used for testing is selected based on a designed policy using validation accuracy information after training for a certain amount of epochs.

Architecture 3: Shallow-CNN-GRU

$inputs$ = **Input**(shape = (22, 1000))
r_1 = **Reshape**((1, 22, 1000))($inputs$)
c_1 = **Conv2D**((40, kernel = (1, 25), stride = 1))(r_1)
r_2 = **Reshape**((1, 880, 976))(c_1)
c_2 = **Conv2D**((40, kernel = (880, 1), stride = 1))(c_1)
sq_1 = **Activation**(Ksquare)(c_2)
r_3 = **Reshape**((1, 40, 976))(sq_1)
ap_1 = **AveragePool2D**((1, 75), stride = (1, 15))(r_3)
log_1 = **Activation**(Klog)(ap_1)
r_4 = **Reshape**(40, 61)(log_1)
g_1 = **GRU**(units = 128)(r_4)
$outputs$ = **Dense**(4, activation = softmax)(g_1)

Architecture 4: Deep-CNN

$inputs$ = **Input**(shape = (22, 1000))
r_1 = **Reshape**((1, 22, 1000))($inputs$)
c_1 = **Conv2D**((25, kernel = (1, 10), stride = 1))(r_1)
c_1 = **Batchnorm2D**()(c_1)
c_1 = **Dropout**(0.5)(c_1)
r_2 = **Reshape**((1, 550, 991))(c_1)
c_2 = **Conv2D**((25, kernel = (550, 1), stride = 1))(r_1)
c_2 = **Batchnorm2D**()(c_2)
c_2 = **Dropout**(0.5)(c_2)
m_1 = **MaxPool2D**((1, 3), stride = (1, 3))(c_2)
r_3 = **Reshape**((1, 25, 330))(m_1)
c_3 = **Conv2D**((50, kernel = (25, 10), stride = 1))(r_3)
c_3 = **Batchnorm2D**()(c_3)
c_3 = **Dropout**(0.5)(c_3)
m_2 = **MaxPool2D**((1, 3), stride = (1, 3))(c_3)
r_4 = **Reshape**((1, 50, 107))(m_2)
c_4 = **Conv2D**((100, kernel = (50,10), stride = 1))(r_4)
c_4 = **Batchnorm2D**()(c_4)
c_4 = **Dropout**(0.5)(c_4)
m_3 = **MaxPool2D**((1, 3), stride = (1, 3))(c_4)
r_5 = **Reshape**((1, 100, 32))(m_3)
c_5 = **Conv2D**((100, kernel = (50,10), stride = 1))(r_5)
c_5 = **Batchnorm2D**()(c_5)
c_5 = **Dropout**(0.5)(c_5)
m_4 = **MaxPool2D**((1, 3), stride = (1, 3))(c_5)
f_1 = **Flatten**()(m_4)
$outputs$ = **Dense**(4, activation = softmax)(f_1)

Architecture 5: LSTM

$inputs$ = **Input**(shape = (22, 1000))
l_1 = **LSTM**(128)($inputs$)
$outputs$ = **Dense**(4, activation = softmax)(l_1)

Architecture 6: GRU

$inputs$ = **Input**(shape = (22, 1000))
g_1 = **GRU**(128)($inputs$)
$outputs$ = **Dense**(4, activation = softmax)(g_1)

Architecture 7: CNN-U

$inputs$ = **Input**(shape = (22, 1000))
r_1 = **Reshape**((22, 1000, 1))($inputs$)
c_1 = **Conv2D**((8, (3, 3), stride = 1,pad = same))(r_1)
c_1 = **Batchnorm2D**()(c_1)
c_1 = **Dropout**(0.5)(c_1)
ap_1 = **AveragePool2D**((2, 2), stride = (1, 1))(c_1)
c_2 = **Conv2D**((16,(3,3), stride = 1,pad = same))(ap_1)
c_2 = **Batchnorm2D**()(c_2)
c_2 = **Dropout**(0.5)(c_2)
c_3 = **Conv2D**((8, (3, 3), stride = 1,pad = same))(c_2)
c_3 = **Batchnorm2D**()(c_3)
c_3 = **Dropout**(0.5)(c_3)
up_1 = **Upsample2D**(size = (2, 2))(c_3)
ct_1 = **Concatenate**()([c_1,up_1])
c_4 = **Conv2D**((4, (3, 3), stride = 1,pad = same))(ct_1)
c_4 = **Batchnorm2D**()(c_4)
c_4 = **Dropout**(0.5)(c_4)
c_5 = **Conv2D**((1, (3, 3), stride = 1,pad = same))(c_4)
c_5 = **Batchnorm2D**()(c_5)
c_5 = **Dropout**(0.5)(c_5)
f_1 = **Flatten**()(c_5)
$outputs$ = **Dense**(4, activation = softmax)(f_1)

4. Results

4.1. Normalization

We compare the result of min-max normalization, z-normalization and no normalization on Shallow-CNN based on the assumption that data normalization scheme will have a similar performance across all architectures. From Table 1, we notice that normalization seems not improving the model's performance as we observe a similar level of test accuracy. However, from Fig. 1, we notice that no normalization will result in a large fluctuation in both training and validation accuracy. In addition, min-max normalization seems to be more effective than the z-normalization. Thus, we select min-max normalization and utilize it in the following pipeline.

4.2. Data augmentation

Based on the same assumption with the above section, we compare the result of noise injection and random cropping using sliding window with original data on Shallow-CNN. The data are processed using min-max normalization.

Architecture 8: CNN-U-LSTM

$inputs$ = **Input**(shape = (22, 1000))
r_1 = **Reshape**((22, 1000, 1))($inputs$)
c_1 = **Conv2D**((8, (3, 3), stride = 1,pad = same))(r_1)
c_1 = **Batchnorm2D**()(c_1)
c_1 = **Dropout**(0.5)(c_1)
ap_1 = **AveragePool2D**((2, 2), stride = (1, 1))(c_1)
c_2 = **Conv2D**((16,(3,3), stride = 1,pad = same))(ap_1)
c_2 = **Batchnorm2D**()(c_2)
c_2 = **Dropout**(0.5)(c_2)
c_3 = **Conv2D**((8, (3, 3), stride = 1,pad = same))(c_2)
c_3 = **Batchnorm2D**()(c_3)
c_3 = **Dropout**(0.5)(c_3)
up_1 = **Upsample2D**(size = (2, 2))(c_3)
ct_1 = **Concatenate**()([c_1,up_1])
c_4 = **Conv2D**((4, (3, 3), stride = 1,pad = same))(ct_1)
c_4 = **Batchnorm2D**()(c_4)
c_4 = **Dropout**(0.5)(c_4)
c_5 = **Conv2D**((1, (3, 3), stride = 1,pad = same))(c_4)
c_5 = **Batchnorm2D**()(c_5)
c_5 = **Dropout**(0.5)(c_5)
l_1 = **LSTM**(units = 128)(c_5)
$outputs$ = **Dense**(4, activation = softmax)(f_1)

Architecture 9: CNN-U-GRU

$inputs$ = **Input**(shape = (22, 1000))
r_1 = **Reshape**((22, 1000, 1))($inputs$)
c_1 = **Conv2D**((8, (3, 3), stride = 1,pad = same))(r_1)
c_1 = **Batchnorm2D**()(c_1)
c_1 = **Dropout**(0.5)(c_1)
ap_1 = **AveragePool2D**((2, 2), stride = (1, 1))(c_1)
c_2 = **Conv2D**((16,(3,3), stride = 1,pad = same))(ap_1)
c_2 = **Batchnorm2D**()(c_2)
c_2 = **Dropout**(0.5)(c_2)
c_3 = **Conv2D**((8, (3, 3), stride = 1,pad = same))(c_2)
c_3 = **Batchnorm2D**()(c_3)
c_3 = **Dropout**(0.5)(c_3)
up_1 = **Upsample2D**(size = (2, 2))(c_3)
ct_1 = **Concatenate**()([c_1,up_1])
c_4 = **Conv2D**((4, (3, 3), stride = 1,pad = same))(ct_1)
c_4 = **Batchnorm2D**()(c_4)
c_4 = **Dropout**(0.5)(c_4)
c_5 = **Conv2D**((1, (3, 3), stride = 1,pad = same))(c_4)
c_5 = **Batchnorm2D**()(c_5)
c_5 = **Dropout**(0.5)(c_5)
g_1 = **GRU**(units = 128)(c_5)
$outputs$ = **Dense**(4, activation = softmax)(f_1)

From Table 2, we notice that random crop is not effective to augment the data although they demonstrate impressive performance in many image processing applications. One pos-

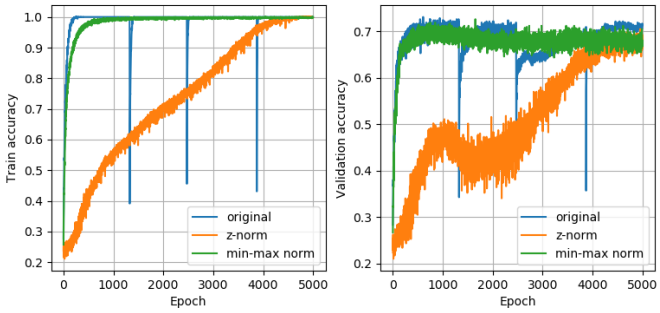

Figure 1. Train and validation accuracy comparison using various normalization schemes on Shallow-CNN in 5000 epochs.

Table 1. Test accuracy of different normalization schemes on Shallow-CNN with 5000 epochs.

Epoch	original	z-norm	min-max norm
1000	67.72%	61.85%	43.12%
2000	67.72%	61.40%	58.01%
3000	67.72%	61.40%	62.53%
4000	67.72%	61.40%	66.14%
5000	67.72%	61.40%	**68.40%**

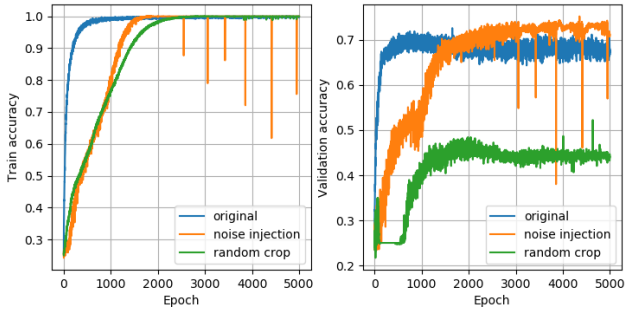

Figure 2. Train and validation accuracy comparison using various data augmentation schemes on Shallow-CNN in 5000 epochs.

sible explanation is that the temporal data is highly correlated. When we crop the input sequence, it is hard to reconstruct the missing information using upsampling and these information play a important role in the extracted features. Noise injection seems to achieve a higher validation accuracy compared with original data but they also result in a high fluctuation, which is similar to the no normalization in the previous section. Thus, we continue to use the original data after normalization without data augmentation in the following sections.

4.3. Single subject optimization

We trained each implemented architecture using extracted single subject and report its test accuracy with 1000 epochs. The models are trained with min-max normalization without data augmentation. All subjects accuracy is calculated as the weighted average of individual single subject accuracy. Shallow-CNN will be shorten as S-CNN in the following paper for simplicity. As shown in Table 3, S-CNN and S-CNN-GRU demonstrate a comparable performance with a 42.89% and 45.82%. We also observe a similar validation accuracy in the training stage. Different from the conclusion made in [3], we notice a drop in test accuracy after adding a RNN layer after S-CNN. Adding a LSTM layer demonstrates a larger drop than adding a GRU layer. One possible explanation for the drop is that LSTM has more parameters than GRU. Because the data used for single subject training is very limited, the model will quickly overfit the training data.

Table 2. Test accuracy of different data augmentation schemes on Shallow-CNN with 5000 epochs.

Epoch	original	noise injection	random crop
1000	43.12%	51.69%	47.40%
2000	58.01%	68.17%	54.63%
3000	62.53%	68.17%	54.63%
4000	66.14%	68.17%	54.63%
5000	**68.40%**	68.17%	55.98%

4.4. All subjects optimization

As shown in Table 4, we evaluate our model's performance after training the models in 1000 epochs with a min-max normalization without data augmentation. Each model selected for testing is based on the best validation accuracy across 1000 epochs. We report the test accuracy for individual subject to compare with the results from optimizing for one single subject. From the table, we notice that the test accuracy for most models are more optimal compared with optimizing with a single subject. This conforms the consensus in deep learning that having more data will generally improve the generalization performance. We also observe that S-CNN and S-CNN-GRU illustrate good performance in both single subject and all subjects optimization. A similar trend is also observed in validation accuracy. Thus, they are selected as candidates for fine tuning.

4.5. Parameter tuning

We train our selected candidates using several hyper-parameter tunning schemes including tuning learning rate, depth of S-CNN, kernel and step size of convolution layer etc. After several trials, we found that extending the training epochs to 10000 is very effective to improve the validation accuracy. From Fig. 3, we can see that S-CNN demonstrates a better performance than S-CNN-GRU and we report its test accuracy as our best testing accuracy. The best model is selected from the best validation score in S-CNN across all epochs. After testing our best model, we achieve a accuracy of **72.01**%.

Table 3. Test accuracy of implemented models trained using single subject with 1000 epochs.

Models	Subject 1	Subject 2	Subject 3	Subject 4	Subject 5	Subject 6	Subject 7	Subject 8	Subject 9	All Subjects
S-CNN	56.00%	30.00%	42.00%	28.00%	31.91%	40.82%	64.00%	30.00%	63.83%	***42.89%***
Deep-CNN	36.00%	32.00%	34.00%	24.00%	29.79%	24.49%	30.00%	14.00%	29.79%	28.22%
LSTM	40.00%	12.00%	24.00%	30.00%	31.91%	22.45%	28.00%	16.00%	31.91%	26.19%
GRU	36.00%	44.00%	40.00%	16.00%	25.53%	44.90%	38.00%	32.00%	27.66%	33.86%
S-CNN-LSTM	40.00%	20.00%	32.00%	42.00%	46.81%	30.61%	42.00%	30.00%	51.06%	37.02%
S-CNN-GRU	52.00%	34.00%	58.00%	30.00%	51.06%	38.78%	60.00%	34.00%	55.32%	***45.82%***
CNN-U	22.00%	38.00%	28.00%	16.00%	29.79%	34.69%	34.00%	18.00%	36.17%	28.44%
CNN-U-LSTM	36.00%	20.00%	26.00%	20.00%	23.40%	26.53%	32.00%	24.00%	25.53%	25.96%
CNN-U-GRU	40.00%	38.00%	30.00%	40.00%	23.40%	26.53%	40.00%	38.00%	29.79%	34.09%

Models and its accuracy are bolded, tilted and underlined if the accuracy is over 40% in both single subject and all subject optimization.

Table 4. Test accuracy of implemented models trained using all subjects with 1000 epochs.

Models	Subject 1	Subject 2	Subject 3	Subject 4	Subject 5	Subject 6	Subject 7	Subject 8	Subject 9	All Subjects
S-CNN	28.00%	36.00%	50.00%	56.00%	51.06%	42.86%	46.00%	36.00%	42.55%	***43.12%***
Deep-CNN	40.00%	40.00%	70.00%	38.00%	51.06%	40.82%	48.00%	50.00%	38.30%	**46.28%**
LSTM	32.00%	26.00%	32.00%	30.00%	40.43%	34.69%	26.00%	18.00%	27.66%	**29.57%**
GRU	36.00%	28.00%	30.00%	30.00%	17.02%	34.69%	34.00%	24.00%	19.15%	28.22%
S-CNN-LSTM	36.00%	24.00%	46.00%	42.00%	44.68%	46.94%	36.00%	42.00%	38.30%	**39.50%**
S-CNN-GRU	44.00%	46.00%	58.00%	54.00%	61.70%	46.94%	60.00%	46.00%	53.19%	***52.14%***
CNN-U	38.00%	48.00%	48.00%	34.00%	59.57%	40.82%	58.00%	42.00%	42.55%	**45.60%**
CNN-U-LSTM	34.00%	38.00%	36.00%	22.00%	38.30%	26.53%	36.00%	26.00%	19.15%	**30.70%**
CNN-U-GRU	38.00%	40.00%	34.00%	22.00%	27.66%	38.78%	24.00%	32.00%	27.66%	31.60%

Models and its accuracy are bolded if they achieve a better accuracy than single subject optimization. Models and its accuracy are tilted and underlined if the accuracy is over 40% in both single subject and all subject optimization.

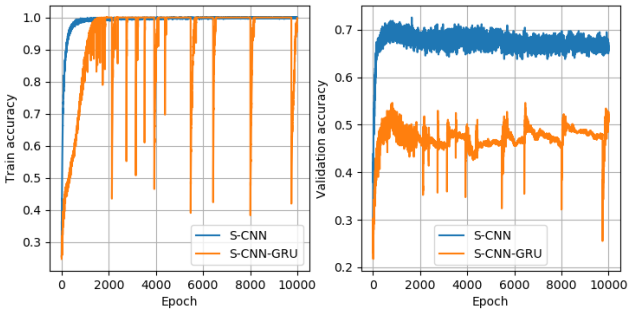

Figure 3. Train and validation accuracy comparison of best model candidates in 10000 epochs.

Table 5. Test accuracy of selected tuning models trained using all subjects with 10000 epochs.

Epoch	S-CNN-GRU	S-CNN
2000	44.24%	58.01%
4000	46.95%	66.14%
6000	46.95%	68.17%
8000	46.95%	71.33%
10000	46.95%	**72.01%**

5. Discussion

In the project, we summarize several observations and provide possible explanations in the following. These tips might be useful in general to design and select a good neural network for different applications.

Observation 1. *If most parameters are on the fully connected layer, the network is likely to encode all the training samples and overfits.*

Explanation. A single fully connected layer contains much more parameters than a convolution layer. In our case, we only have 2115 samples for training and validation. The model tends to remember the data if the size of parameters is too large.

Observation 2. *Selecting the model with best validation accuracy might not be optimal to select the best architecture. Discovering the trend in the model is more helpful.*

Explanation. When we are selecting the best architecture, we generally choose the epoch with best validation score, assuming it as the representative of the architecture, and compare it across different candidates. However, if the model is not robust enough, it will have large fluctuations in both the training and validation accuracy although with a high best validation accuracy. Its generalization ability is also limited. Plotting the validation accuracy across the entire epochs in the training process is more helpful to discover the trend and select the effective models instead of blindly comparing the best validation accuracy.

There are still some possibilities to further improve the model's performance. Due to the limit of time, we only tried two data augmentation schemes and showed that they are not suitable for our design. There are more options which

might be useful. One option is to use the variational autoencoder to select the most representative features and generate more samples. The other option is to train a generative adversarial network to learn the features of dataset and create more samples. These could be effective and are worth exploring.

References

[1] O. Ronneberger, P. Fischer, and T. Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, pages 234–241. Springer, 2015.

[2] R. T. Schirrmeister, J. T. Springenberg, L. D. J. Fiederer, M. Glasstetter, K. Eggensperger, M. Tangermann, F. Hutter, W. Burgard, and T. Ball. Deep learning with convolutional neural networks for brain mapping and decoding of movement-related information from the human eeg. arxiv, 2017. *arXiv preprint arXiv:1703.05051*.

[3] J. Wang, Y. Yang, J. Mao, Z. Huang, C. Huang, and W. Xu. Cnn-rnn: A unified framework for multi-label image classification. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2285–2294, 2016.

[4] X. Zhang, D. G. Martin, M. Noga, and K. Punithakumar. Fully automated left atrial segmentation from mr image sequences using deep convolutional neural network and unscented kalman filter. In *2018 IEEE International Conference on Bioinformatics and Biomedicine (BIBM)*, pages 2316–2323. IEEE, 2018.