Protein Secondary Structure Prediction - A Comprehensive Comparison

Noble Kennamer[1+] **and Wentao Zhu**[1+]

[1]Department of Computer Science, University of California, Irvine
[1]{nkenname, wentaoz1}@uci.edu

ABSTRACT

In this project, we employ different models, such as logistic regression, logistic regression with sliding window of different neighbors, Hidden Markov Model (HMM), Structured Perceptron (Conditional Random Field, CRF), Convolutional Neural Network (CNN), and Cascaded CNN and Gated Recurrent Unite (GRU), to do the protein secondary structure prediction. We evaluate these methods use precision and recall for each structure, and accuracy as the metrics for the comprehensive comparison on the problem. Further work can be done based on the project to get the best performance for protein secondary structure prediction.

Introduction

Protein secondary structure prediction is an conventional and important problem in bioinformatics. For example, drug design requires understanding of protein structures[1]. The protein second structure prediction is an essential intermediate step for the protein structure understanding.

Protein secondary structure prediction is a well known problem in both biology and machine learning societies. There are a lot of Statistical methods used for the problem[2]. However, due to ineffective features, these methods only obtain very low performance even on the Q3 task, three class classification, helix (H), strand (E) and coil (C). Evolutionary information of proteins and position specific scoring matrices are used to improve the protein secondary structure prediction performance on the Q3 task[3,4]. But these methods face great difficulty when applied to Q8 task, eight class classification, 3_{10}-helix (G), α-helix (H), π-helix (I), β-strand (E), β-bridge (B), β-turn (T), bend (S) and loop or irregular (L). Artificial neural networks also have been applied to the problem, such as support vector machines (SVM)[5], recurrent neural networks (RNN)[6], conditional neural fields[7], and generative stochastic networks[8].

In this project, we compare various methods, such as logistic regression, logistic regression with different windows, HMM, CRF, CNN and combined CNN and GRU models on the CB6133 dataset produced with PISCES CullPDB[9].

Methods

We attempted a variety of different methods for the task of secondary protein structure prediction. Logistic regression was used to establish a baseline approach. For logistic regression we used the the profiles as our features. Note that this method does not take into account any structure information we are simply trying to classify the profiles to the correct structure label. We then extended logistic regression by running logistic regression over sliding windows of the seqeuences. Essentially what this does is augment the profile of an amino acid with the profiles of of its neartest neighbor. We used this approach with the two neartest neighbors (one of each side) and its four nearest neighbors to on each side. Note that we used the implementation of logistic regression from sklearn to train these models.

We also used two different sequence models, a hidden markov model and a structured perceptron. The hidden markov model models the sequence using the markov assumption that the current input is only dependent on the previous input and this assumption also for the efficient construction of conditional probability distributions. The structured perceptron is a similiar model and is implemented as a linear-chain conditional random field and the peceptron algorithm is used to train a classifier for the structure label. For both of these models we used an implementation from seqlearn.

We also used several different neural network models. We used a fairly simple convolutional neural network that emploeted a sliding window approach over the sequences. The CNN was using a window size of 41 (the 20 nearest neighbors on each side of the current profile) with two convolutional layers and two fully connected layer and using the cross entropy cost function. The first convolutional layer consistend of 30 feature maps and was a kernal size of 5 by 22 with rectified linear units for the activation function. The second convolutional layer also had 30 feature maps and a kernal size of 5 by 22 with rectified linear

units. The first fully connected was 1000 rectified linear units and this was then fed into the final fully connected layer of 8 inputs that computed the cost. This network was implemented using Tensorflow and trained on two Nvidia Telska K20s.

We reprodue the best reported method, cascade CNN and GRU network[10]. We implement the deep network using Keras package with Theano backend. The network structure is illustrated in Fig. 1.

The model firstly converts 21 dimensional one hot amino acid representation into a 50 dimension continuous representation using a feedforward embedding layer. Then concatenate the 50 dimensional embedded features with the 21 dimensional profile features. After that, input these 71 dimensional features into three independent CNNs with kernel sizes of $3, 7, 11$ respectively and concatenate these CNN features together. They call the deep CNN features as local feature. Next, the model inputs these local features into three bidirectional GRU layers. The model called the last layer's GRU features as global features. Concatenate the global features with local features to get a complete representation. After that, the model employs two fully connected layer to learn more invariance features. Lastly, multi-task learning is used for the prediction layer. The second task in the multi-task learning is relative and absolute solvent accessibility.

The number of CNN kernels in each CNN is 64. The dimension of local features is 192. Each GRU has 300 units. Dropout is used for GRU with dropout probability of 0.5. The dimension of complete features are 792. The fully connected layer has 200 hidden units. The multi-task cost function is cross entropy and the trade off the two tasks' cost functions is 1.

The following is our experience in the reproduce of the work. Firstly, for Dropout of GRU, it is better to only dropout W and not use any other regularization for the GRU. Secondly, we need to add a masking scheme in the accuracy function, because the 1D convolutional operator does not support mask layer in the input. The varied length of sequences requires the masking. The easiest way to do the masking in the accuracy function is as

$$Accuracy = \frac{\sum_n \left(argmax(y_n) == argmax(\hat{y_n})\right) \times sum(y_n)}{\sum_n y_n}, \tag{1}$$

where $\hat{y_n}$ is the predicted label, and y_n is the true label. A good point is that, we do not need to change the loss function, because the loss function for cross entropy is $\sum_n \sum_i y_{n,i} log(p(y_{n,i}))$. For the blank states, all the $y_{n,i}$ are zeros, so they are not counted into the loss function. Thirdly, to tune the parameters, it is really a good way to over-fit the model first, and add more regularization later. Fourthly, it is better to use no regularization for the embedding layer and the output layer. Fifthly, we use L_2 regularization with factor of 1×10^{-3} in the CNN layers and the two fully connected layers. Sixthly, due to the memory issues, we use the batchsize of 96. The tuned learning rate is 3×10^{-4} for Adam[11]. The number of epochs are 100. It costs about 7 hours to train a model running the 100 epochs. We obtain the result in Fig. 2.

Results

For all models we will present the precision and recall of each structure label and the total Q8 accuracy.

Logistic Regression

For logistic regression the precision and recall results are in Table 1. And we got 0.45 accuracy on the CB6133 dataset for the Q8 task.

Categories	Precision	Recall
L	0.35	0.35
B	-	0.0
E	0.46	0.45
G	-	0.0
I	-	0.0
H	0.5	0.75
S	0.36	0.0008
T	0.36	0.26

Table 1. The precision and recall of Logistic Regression.

For logistic regression with a sliding window of 3 the precision and recall results are in Table 2. And we got 0.55 accuracy on the CB6133 dataset for the Q8 task.

For logistic regression with a sliding window of 5 the precision and recall results are in Table 3. And we got 0.58 accuracy on the CB6133 dataset for the Q8 task.

Categories	Precision	Recall
L	0.44	0.5
B	-	0.0
E	0.58	0.60
G	0.32	0.0004
I	-	0.0
H	0.62	0.79
S	0.34	0.04
T	0.41	0.39

Table 2. The precision and recall of logistic regression with a sliding window of 3.

Categories	Precision	Recall
L	0.46	0.54
B	-	0.0
E	0.62	0.65
G	0.23	0.01
I	-	0.0
H	0.67	0.81
S	0.36	0.07
T	0.42	0.40

Table 3. The precision and recall of logistic regression with a sliding window of 5.

Categories	Precision	Recall
L	1.0	0.04
B	-	0.0
E	-	0.0
G	-	0.0
I	-	0.0
H	0.34	1.0
S	-	0.0
T	-	0.0

Table 4. The precision and recall of HMM.

Hidden Markov Model and Structured Perceptron

For the hmm the precision and recall results are in Table 4. And we got 0.35 accuracy on the CB6133 dataset for the Q8 task.

The precision and recall results for the structured perceptron are in Table 5. And we got 0.45 accuracy on the CB6133 dataset for the Q8 task.

Categories	Precision	Recall
L	0.29	0.43
B	0.015	0.014
E	0.47	0.71
G	0.15	0.001
I	-	0.0
H	0.83	0.50
S	0.18	0.69
T	0.25	0.32

Table 5. The precision and recall of the Structured Perceptron.

Convolutional Neural Network with sliding Window

For the convolutional neural network with sliding window the precision and recall results are in Table 6. And we got 0.79 accuracy on the CB6133 dataset for the Q8 task.

Categories	Precision	Recall
L	0.67	0.72
B	.62	0.40
E	0.86	0.87
G	0.69	0.65
I	1.0	1.0
H	0.91	0.95
S	0.56	0.47
T	0.68	0.62

Table 6. The precision and recall of the CNN.

Reproduce Cascade CNN and GRU network

We calculate the precision and recall values for each class as in Table 7. And we got 0.7327 accuracy on the CB6133 dataset for the Q8 task.

Categories	Precision	Recall
H	0.8775	0.9345
E	0.8044	0.8557
L	0.5819	0.6736
T	0.5854	0.5671
S	0.5170	0.2390
G	0.4048	0.3409
B	0.3333	0.0104
I	0.0000	0.0000

Table 7. The precision and recall of the reproduced model.

The performance in Table 7 is consistent with the result in[10], which is better than the result in[8]. The accuracy curve with the epochs are shown in Fig. 3.

Discussion

From the results above we can see what we were able to obtain the best performance using a Convolutional Neural Network with sliding windows. Note that this model only focuses on modelling local structure to predict the secondary structure label. This is at odds with the state of the art models that model both the local and global structure. The fact that our CNN does even better seems to indicate that while global structure might be importnat the most importnat factor for achieving good performance is modelling the local structure as well as possible. We would argue that our model is better able to model local structure and this is what allows it to achieve higher performance. We believe that by extending our model to also include global structure could result in even better results and is an approach we will try in the future.

Another aspect that was surprising was how poorly the hidden markov model and structured perceptron performed. Both of these results got worst performance than basic logistic regression. Both of these models attempt to model the sequence as a linear chain where the current factor (distribution) only depends on the previous input. It is clear from the results that this assumption is too restictive to accuractely model the data.

Ultimately our main takeaway from this project is that modelling the local structure is the most important aspect for getting good accuracy and adding global structure would be useful to increase that accuracy by some amount.

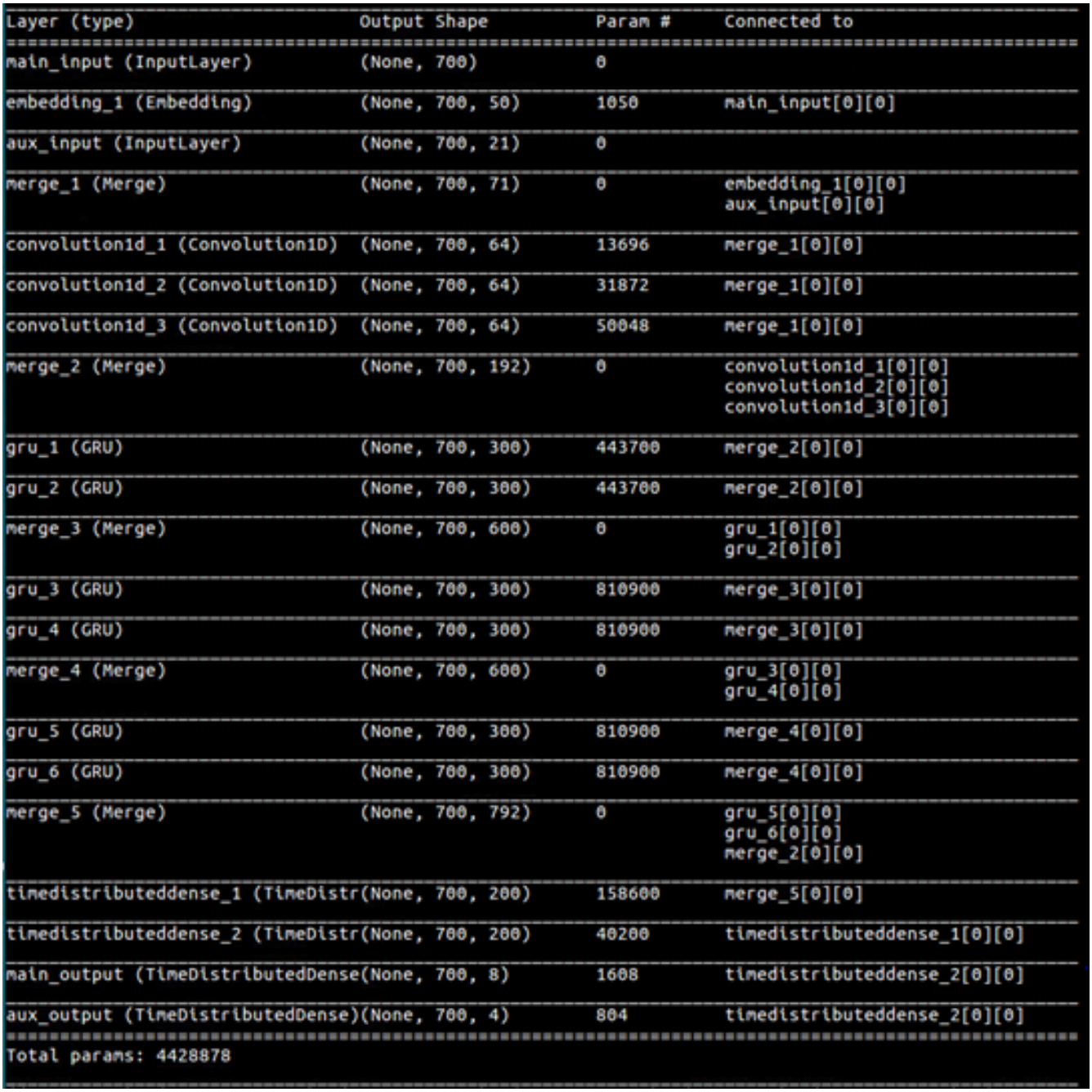
References

1. Noble, M. E., Endicott, J. A. & Johnson, L. N. Protein kinase inhibitors: insights into drug design from structure. *Science* **303**, 1800–1805 (2004).

2. Chou, P. Y. & Fasman, G. D. Prediction of protein conformation. *Biochemistry* **13**, 222–245 (1974).
3. Rost, B. & Sander, C. Prediction of protein secondary structure at better than 70% accuracy. *Journal of molecular biology* **232**, 584–599 (1993).
4. Jones, D. T. Protein secondary structure prediction based on position-specific scoring matrices. *Journal of molecular biology* **292**, 195–202 (1999).
5. Hua, S. & Sun, Z. A novel method of protein secondary structure prediction with high segment overlap measure: support vector machine approach. *Journal of molecular biology* **308**, 397–407 (2001).
6. Pollastri, G., Przybylski, D., Rost, B. & Baldi, P. Improving the prediction of protein secondary structure in three and eight classes using recurrent neural networks and profiles. *Proteins: Structure, Function, and Bioinformatics* **47**, 228–235 (2002).
7. Wang, Z., Zhao, F., Peng, J. & Xu, J. Protein 8-class secondary structure prediction using conditional neural fields. *Proteomics* **11**, 3786–3792 (2011).
8. Zhou, J. & Troyanskaya, O. G. Deep supervised and convolutional generative stochastic network for protein secondary structure prediction. In *ICML*, 745–753 (2014).
9. Wang, G. & Dunbrack, R. L. Pisces: a protein sequence culling server. *Bioinformatics* **19**, 1589–1591 (2003).
10. Li, Z. & Yu, Y. Protein secondary structure prediction using cascaded convolutional and recurrent neural networks. *arXiv preprint arXiv:1604.07176* (2016).
11. Kingma, D. & Ba, J. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).

Acknowledgements (not compulsory)

The authors would like to thank NVIDIA for graphic card donation in Prof. Xiaohui Xie's Lab.



Layer (type)	Output Shape	Param #	Connected to
main_input (InputLayer)	(None, 700)	0	
embedding_1 (Embedding)	(None, 700, 50)	1050	main_input[0][0]
aux_input (InputLayer)	(None, 700, 21)	0	
merge_1 (Merge)	(None, 700, 71)	0	embedding_1[0][0] aux_input[0][0]
convolution1d_1 (Convolution1D)	(None, 700, 64)	13696	merge_1[0][0]
convolution1d_2 (Convolution1D)	(None, 700, 64)	31872	merge_1[0][0]
convolution1d_3 (Convolution1D)	(None, 700, 64)	50048	merge_1[0][0]
merge_2 (Merge)	(None, 700, 192)	0	convolution1d_1[0][0] convolution1d_2[0][0] convolution1d_3[0][0]
gru_1 (GRU)	(None, 700, 300)	443700	merge_2[0][0]
gru_2 (GRU)	(None, 700, 300)	443700	merge_2[0][0]
merge_3 (Merge)	(None, 700, 600)	0	gru_1[0][0] gru_2[0][0]
gru_3 (GRU)	(None, 700, 300)	810900	merge_3[0][0]
gru_4 (GRU)	(None, 700, 300)	810900	merge_3[0][0]
merge_4 (Merge)	(None, 700, 600)	0	gru_3[0][0] gru_4[0][0]
gru_5 (GRU)	(None, 700, 300)	810900	merge_4[0][0]
gru_6 (GRU)	(None, 700, 300)	810900	merge_4[0][0]
merge_5 (Merge)	(None, 700, 792)	0	gru_5[0][0] gru_6[0][0] merge_2[0][0]
timedistributeddense_1 (TimeDistr	(None, 700, 200)	158600	merge_5[0][0]
timedistributeddense_2 (TimeDistr	(None, 700, 200)	40200	timedistributeddense_1[0][0]
main_output (TimeDistributedDense	(None, 700, 8)	1608	timedistributeddense_2[0][0]
aux_output (TimeDistributedDense)	(None, 700, 4)	804	timedistributeddense_2[0][0]

Total params: 4428878

Figure 1. The network structure of the cascade CNN and GRU network.

```
Epoch 90/100
Epoch 00089: val_main_output_weighted_accuracy did not improve
278s - loss: 0.3908 - main_output_loss: 0.2071 - aux_output_loss: 0.1721 - main_output_weighted_accuracy: 0.7449 - aux_output_weighted_accuracy: 0.7713 - val_loss: 0.3888 - val_
main_output_loss: 0.2191 - val_aux_output_loss: 0.1697 - val_main_output_weighted_accuracy: 0.7301 - val_aux_output_weighted_accuracy: 0.7737
Epoch 91/100
Epoch 00090: val_main_output_weighted_accuracy improved from 0.73090 to 0.73192, saving model to ./ijcaibestacctwomasknowunoembedsoftmaxregular3e-4.h5
278s - loss: 0.3902 - main_output_loss: 0.2066 - aux_output_loss: 0.1719 - main_output_weighted_accuracy: 0.7449 - aux_output_weighted_accuracy: 0.7713 - val_loss: 0.3882 - val_
main_output_loss: 0.2176 - val_aux_output_loss: 0.1706 - val_main_output_weighted_accuracy: 0.7319 - val_aux_output_weighted_accuracy: 0.7727
Epoch 92/100
Epoch 00091: val_main_output_weighted_accuracy did not improve
277s - loss: 0.3908 - main_output_loss: 0.2072 - aux_output_loss: 0.1720 - main_output_weighted_accuracy: 0.7448 - aux_output_weighted_accuracy: 0.7715 - val_loss: 0.3886 - val_
main_output_loss: 0.2191 - val_aux_output_loss: 0.1695 - val_main_output_weighted_accuracy: 0.7282 - val_aux_output_weighted_accuracy: 0.7741
Epoch 93/100
Epoch 00092: val_main_output_weighted_accuracy improved from 0.73192 to 0.73232, saving model to ./ijcaibestacctwomasknowunoembedsoftmaxregular3e-4.h5
278s - loss: 0.3890 - main_output_loss: 0.2058 - aux_output_loss: 0.1716 - main_output_weighted_accuracy: 0.7462 - aux_output_weighted_accuracy: 0.7720 - val_loss: 0.3874 - val_
main_output_loss: 0.2180 - val_aux_output_loss: 0.1694 - val_main_output_weighted_accuracy: 0.7323 - val_aux_output_weighted_accuracy: 0.7742
Epoch 94/100
Epoch 00093: val_main_output_weighted_accuracy did not improve
279s - loss: 0.3881 - main_output_loss: 0.2050 - aux_output_loss: 0.1715 - main_output_weighted_accuracy: 0.7474 - aux_output_weighted_accuracy: 0.7722 - val_loss: 0.3889 - val_
main_output_loss: 0.2194 - val_aux_output_loss: 0.1695 - val_main_output_weighted_accuracy: 0.7306 - val_aux_output_weighted_accuracy: 0.7729
Epoch 95/100
Epoch 00094: val_main_output_weighted_accuracy did not improve
278s - loss: 0.3875 - main_output_loss: 0.2046 - aux_output_loss: 0.1713 - main_output_weighted_accuracy: 0.7476 - aux_output_weighted_accuracy: 0.7724 - val_loss: 0.3865 - val_
main_output_loss: 0.2176 - val_aux_output_loss: 0.1689 - val_main_output_weighted_accuracy: 0.7313 - val_aux_output_weighted_accuracy: 0.7743
Epoch 96/100
Epoch 00095: val_main_output_weighted_accuracy did not improve
278s - loss: 0.3861 - main_output_loss: 0.2035 - aux_output_loss: 0.1711 - main_output_weighted_accuracy: 0.7490 - aux_output_weighted_accuracy: 0.7726 - val_loss: 0.3868 - val_
main_output_loss: 0.2178 - val_aux_output_loss: 0.1690 - val_main_output_weighted_accuracy: 0.7312 - val_aux_output_weighted_accuracy: 0.7738
Epoch 97/100
Epoch 00096: val_main_output_weighted_accuracy did not improve
278s - loss: 0.3860 - main_output_loss: 0.2036 - aux_output_loss: 0.1709 - main_output_weighted_accuracy: 0.7485 - aux_output_weighted_accuracy: 0.7728 - val_loss: 0.3871 - val_
main_output_loss: 0.2181 - val_aux_output_loss: 0.1690 - val_main_output_weighted_accuracy: 0.7322 - val_aux_output_weighted_accuracy: 0.7743
Epoch 98/100
Epoch 00097: val_main_output_weighted_accuracy did not improve
278s - loss: 0.3866 - main_output_loss: 0.2041 - aux_output_loss: 0.1710 - main_output_weighted_accuracy: 0.7481 - aux_output_weighted_accuracy: 0.7730 - val_loss: 0.3883 - val_
main_output_loss: 0.2184 - val_aux_output_loss: 0.1699 - val_main_output_weighted_accuracy: 0.7322 - val_aux_output_weighted_accuracy: 0.7734
Epoch 99/100
Epoch 00098: val_main_output_weighted_accuracy did not improve
278s - loss: 0.3854 - main_output_loss: 0.2033 - aux_output_loss: 0.1706 - main_output_weighted_accuracy: 0.7491 - aux_output_weighted_accuracy: 0.7734 - val_loss: 0.3898 - val_
main_output_loss: 0.2204 - val_aux_output_loss: 0.1694 - val_main_output_weighted_accuracy: 0.7292 - val_aux_output_weighted_accuracy: 0.7729
Epoch 100/100
Epoch 00099: val_main_output_weighted_accuracy improved from 0.73232 to 0.73268, saving model to ./ijcaibestacctwomasknowunoembedsoftmaxregular3e-4.h5
279s - loss: 0.3841 - main_output_loss: 0.2023 - aux_output_loss: 0.1704 - main_output_weighted_accuracy: 0.7502 - aux_output_weighted_accuracy: 0.7738 - val_loss: 0.3874 - val_
main_output_loss: 0.2180 - val_aux_output_loss: 0.1694 - val_main_output_weighted_accuracy: 0.7327 - val_aux_output_weighted_accuracy: 0.7729
```

Figure 2. The result of the cascade CNN and GRU network.

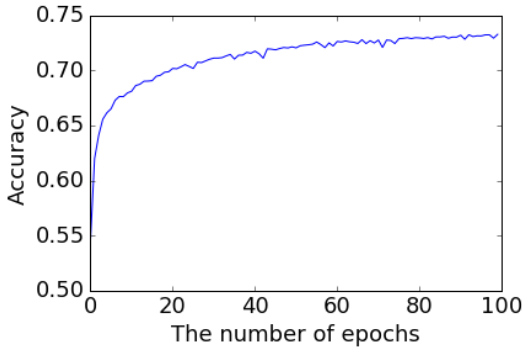


Figure 3. The accuracy curve with the epochs.