Questions and Answers using the SQuAD 2.0 dataset and the QANet architecture

Srinivas Raghavan
sraghava@stanford.edu

Abstract

Significant advances have been made recently in NLP in the area of reading comprehension which is useful in machine translation, question answering, summarization and other NLP tasks. Most of these models use an RNN which, though successful, have well documented downsides. A model, called the QANet architecture using only CNNs and attention (and based on the Transformer architecture) has been proposed that does not suffer from these downsides. This paper implements the QANet architecture and analyses the results of evaluation on the SQuAD 2.0 dataset. In addition, aspects of the transformerXL architecture are added with a goal to further enhance model performance.

1 Introduction

Most models in NLP used for reading comprehension (including question answering) use recurrent neural networks (RNN). The QANet model (Yu et al 2018 [1]) instead, uses convolutional neural networks (CNN) and attention mechanisms like multi-head attention (Vaswani et al 2016 [2]) for self-attention and standard attention (Seo et al 2016 [3]) for C2Q and Q2C attention. The main reason for choosing CNNs over RNNs is the speedup achieved during training.

In addition to the QANet architecture, Yu et al 2018 also proposed a novel idea for training on SQuAD question answering dataset whereby they implemented a data augmentation technique. In this technique, the standard SQuAD dataset is augmented by first translating the dataset to a different language (French in this case) and then back to English. This allows the model to learn different variations of the same sentence. It is shown that this does indeed help to create a better model by using this augmented data for training.

The aim of this paper is to

- Implement the QANet architecture in PyTorch
- Make modifications to add a ‘noAnswer’ option
- Evaluate the performance of the model on the SQuAD 2.0 dataset
- Apply aspects of Transformers XL (Dai, Zihang et al 2019 [4]) and re-evaluate performance

The QANet performance is evaluated against the given baseline which is a modified BIDAF model (Seo et. al 2016 [3]) without the character embedding and with a no-answer prediction extension required for the SQuAD 2.0 dataset.

2 Related Work

Question answering in NLP has wide applications for retrieving information. Traditionally, this has been implemented using separate components like part-of-speech tagging, sentiment analysis and others. In the past few years, neural network models have been shown to be performant for NLP tasks.

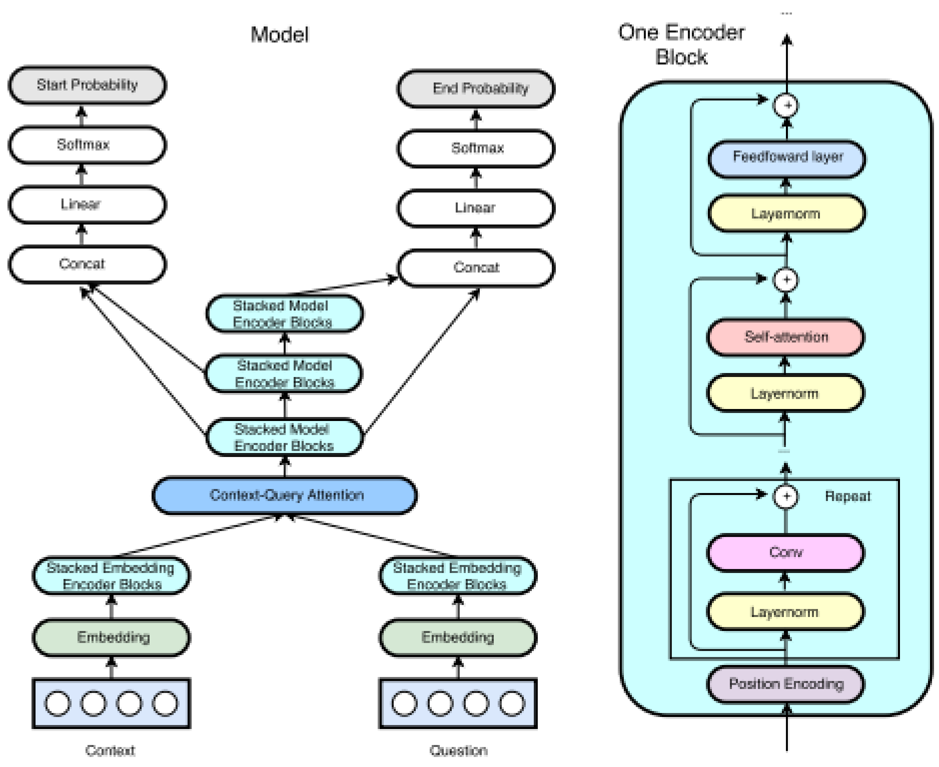

Figure 1: QAnet architecture from [1]

RNNs remain popular in architectures for NLP tasks but a number of efforts have attempted to tackle the vanishing gradient problem ([5]) using weight initialization ([6]) etc. Vaswani et al [2] introduced the transformer which removed dependency on RNNs, opting only for attention mechanisms. The QANet architecture [1] brought back CNNs along with the attention mechanism and the TransformerXL [4] introduced longer term dependency learning with extended context.

Other recent developments include development of a single architecture for multiple tasks like decaNLP [7] which uses the same architecture for question answering, summarization, sentiment analysis etc. Indeed, all these tasks are achieved using the question answering paradigm. Another recent development is the use of pre-trained contextual embeddings like ELMo [8] and BERT [9] that can also be easily adapted for question answering.

3 Model

The model architecture is described here and is depicted in Figure 3.

3.1 Input Embedding Layer

Each word from the context and questions is converted to a vector $v \epsilon R^{500}$ by concatenating a GLoVE word embedding ϵR^{300} and a character embedding ϵR^{200} or, x = [x_w ; x_c]

A two-layer highway network (Srivastava et al 2015 [10]) is used as the final output 'x' of this layer where, $x \epsilon R^{500}$

3.2 Embedding Encoding Layer

This layer consists of one block within which there are 3 types of sub-layers - a convolution layer followed by a self-attention layer followed by a feed-forward layer. The convolution layer is a depthwise separable convolution (Kaiser et al 2017). The self-attention is a multi-head attention mechanism (Vaswani et al 2017 [2]). For each operation ‘f’ (conv/self-attention/ffn), it is performed as f(layernorm(x)) + x (there is a full identity path from input to output of each sub layer. Layernorm is performed as in Ba et al. 2016 [11].

3.3 Context-Query Attention Layer

The matrix $S \epsilon R^{nxm}$ (n = length of context, m = length of query) is constructed. The entry S_{ij} is obtained using trilinear function

$$f(q, c) = W_0[q, c, q \odot c] \tag{1}$$

where $\odot$ indicates elementwise multiplication

Context-To-Query attention

$$A = \overline{S}.Q^T$$

where, $\overline{S} \ = \ softmax(\ row(S) \ )$
Q = query embedding matrix

And, Query-To-Context attention

$$B = \overline{S} \ . \ \overline{\overline{S}}^T \ . \ C^T$$

where, $\overline{\overline{S}} \ = \ softmax(\ column(S) \ )$
C = context embedding matrix

3.4 Model Encoder Layer

There are 3 model encoder $layers \ = \ M_0, \ M_1 \ and \ M_2$, each encoding block is the same as Embedding Encoder block (with some differences in hyperparameters). The input to each block is $[\ c, \ a, \ c \odot a, \ c \odot b \ ]$

3.5 Output Layer

The output layer calculates the following probabilities

$$p_{start} = softmax(W_1[M_0; M_1])$$
$$p_{end} = softmax(W_1[M_0; M_1])$$

where, W_1 and W_2 are learnable weights. Score of a span is the product of its start and end probabilities. The span with the highest score is chosen as the answer. The highest scoring scan is computed as in the baseline.

3.6 Objective Function

Finally, the objective function is

$$L(\theta) = -\frac{1}{N} \sum_{i}^{N} \left[\ log(p^1_{y^1_i}) + log(p^2_{y^2_i}) \ \right] \tag{2}$$

where, y^1_i and y^2_i are respectively, the ground truth for the start and end positions for example i. An Adam optimizer was used, more details in the next section.

3.7 No-answer Prediction

The procedure outlined in Omer Levy et al (2017 [12]) is used. An OOV token is introduced at the start of each context, if $p_{start}(0).p_{end}(0)$ is greater than the predicted answer span then no-answer is predicted.

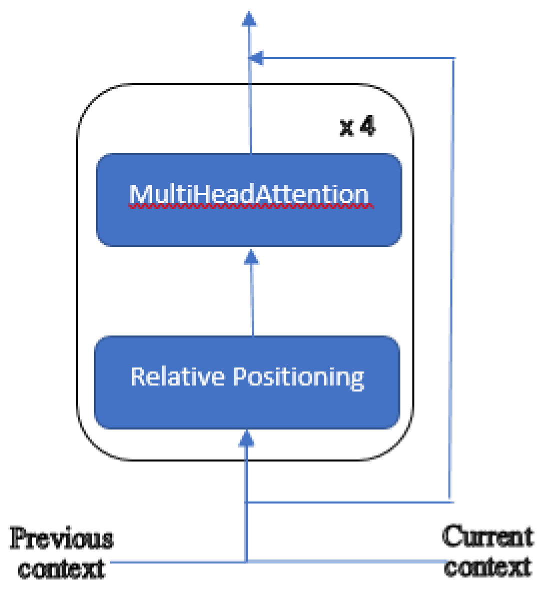

Figure 2: TransformerXL extension showing multihead attention with previous context and relative positioning. Layernorm, dropout etc. not shown

3.8 TransformerXL extension

Dai, Zihang et al 2019 ([4]) introduced the TransformerXL with the ability to use the previous output from each attention layer along with the current input. This allows the attention architecture to use information seen previously when calculating the attention score. To allow this, it is also necessary to have relative positioning.

The QANet implementation was extended by replacing the attention sublayer in each of the encoder blocks in the QAnet architecture. The details of the attention sublayer are shown in Figure 2.

4 Experiments

4.1 Dataset

The SQuAD 2.0 question answering dataset was used (Pranav Rajpurkar et al [13][14]). The official version was slightly modified (details in baseline), consisting of only the train and dev sets. The dev set was pre-split into dev and test sets and the train and dev sets were provided. The context length was set to 400 and question length was set to 50. Prediction length was set at 15.

4.2 Experimental Setup

All hyperparameter values were kept the same as the original QANet paper with the following exceptions –

- The number of heads for multi-attention was 4 instead of 8 because of memory constraints on the GPU
- The <UNK> token was not set to be trainable
- Dropout for every other layer was not implemented
- Batch size is 16 instead of 32 due to memory constraints on the GPU

The hyperparameters can be seen in Table 1

Table 1: Model hyperparameters

Hyperparameter	Original QANet	Current Implementation	with TransformerXL
Embedding Layer			
Glove Word Embedding Size	300	300	Same
Char Embedding Size	200	200	Same
Word Embedding Dropout	0.1	0.1	Same
Char Embedding Dropout	0.05	0.05	Same
Embedding Encoder Layer			
Number of Blocks	1	1	Same
Number of CNN layers	4	4	Same
Kernel Size (CNN)	7	7	Same
Number of filters (CNN)	128	128	Same
Number multi-attention heads	8	4	2
Attention dropout	0.1	0.1	Same
Hidden embed size (d_model)	128	128	Same
Layer dropout	0.1	0.1	Same
Model Encoder Layer			
Number of blocks	7	7	1
Number of CNN layers	2	2	Same
Kernel size(CNN)	5	5	Same
All other parameters same as Embedding Encoder Layer	Same	Same	
For TransformerXL			
Number of Multihead attention layers	N/A	N/A	4
Memory length	N/A	N/A	401 (sentence length)

ADAM optimizer was used (Kingma & Ba, 2014 [15]). Stochastic depth method for layer dropout was used (Huang et al 2016 [16]) as given below

$$p_l = 1 - \frac{1}{L}\big(1 - p_L\big) \tag{3}$$

where L is the last layer and $p_L = 0.9$

Optimizer parameters are shown in Table 2

Table 2: Optimizer Parameters

Parameter	Original Paper	Current Paper
Optimizer Parameters		
L2 weight decay(λ)	3×10^{-7}	3×10^{-7}
$\beta1, \beta2$	0.8,0.9999	0.8,0.999
ϵ	10^{-7}	10^{-7}
EMA decay rate	0.9999	0.9999

All training was done using a 2 GPU machine with a total of 16GB of GPU memory.

4.3 Results

The EM/F1 scores are given in Table 3

Table 3: Results (non-PCE)

	Dev EM	Dev F1	Test EM	Test F1
QANet Score	64.208	67.722	56.230	60.332
QANet With Transformer XL	53.203	58.743	53.609	56.948
Baseline Score	55.991	59.291	56.298	59.920

See figures 3-6.

5 Analysis

The original QANet paper (Yu et. al, 2018) was evaluated on the SQuAD 1.1 dataset. As far as can be seen, there are no published results for SQuAD 2.0 for the QANet architecture. Therefore, as stated previously, the comparison above is with the given baseline.

Progressive improvements in F1 scores was seen as the model was being built incrementally. The optimizer was changed from Adagrad to ADAM. Dropout values were added as indicated in Table 1.

The dev NLL remained almost horizontal after hitting its lowest value unlike the baseline (not shown in figure) which started to increase after reaching bottom. The training NLL decreased consistently throughout.

Looking at the official SQuAD leaderboard it appears the expectation for a single model, non-PCE implementation is to achieve an F1 score of at least 70 (without data augmentation etc.).

5.1 Analysis for QANet implementation

It is seen that the dev F1 score for the QANet architecture was easily higher than the baseline whereas the test F1 score is just a bit higher than the test baseline (see Figure 4). This drastic fall in F1 score from dev to test for the QANet implementation could be caused by the following

- The distribution of the dev and test set could be very different. Note that there was no data augmentation performed on either the training or dev set
- The dev model with the best F1 score was likely picked up from a region where the model had started overfitting the data

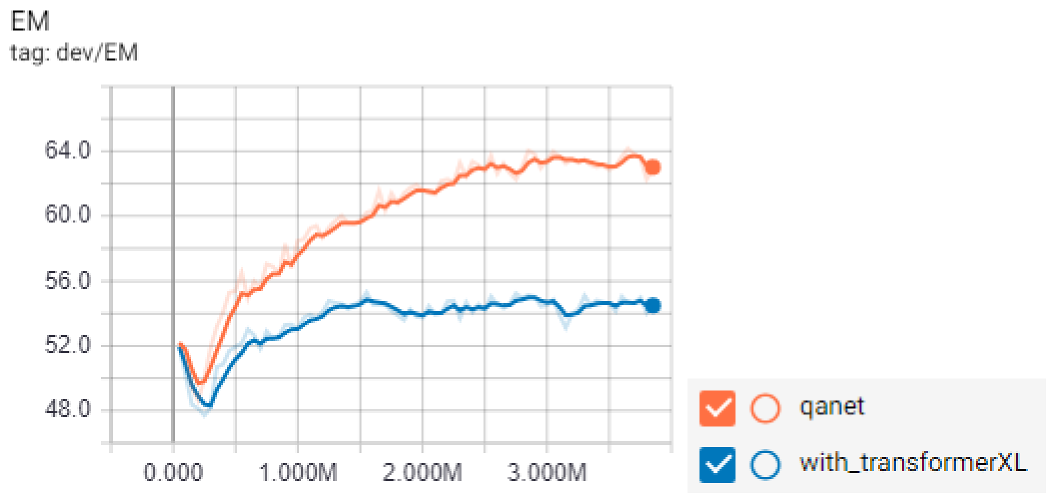

Figure 3: EM

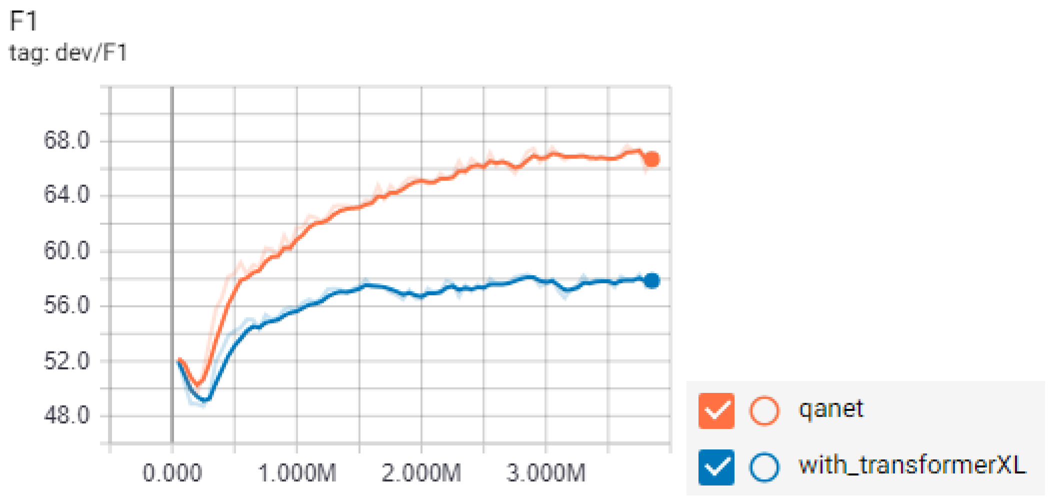

Figure 4: F1

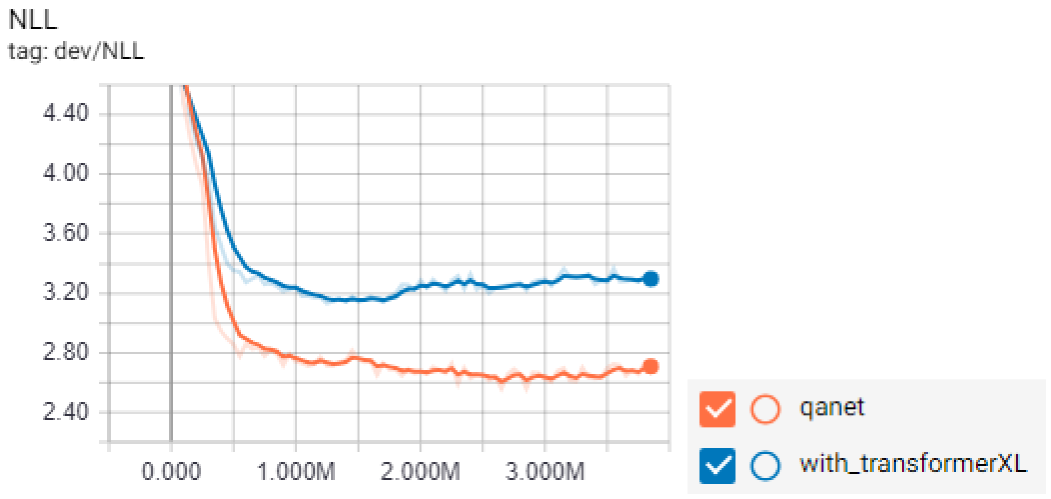

Figure 5: Dev Loss

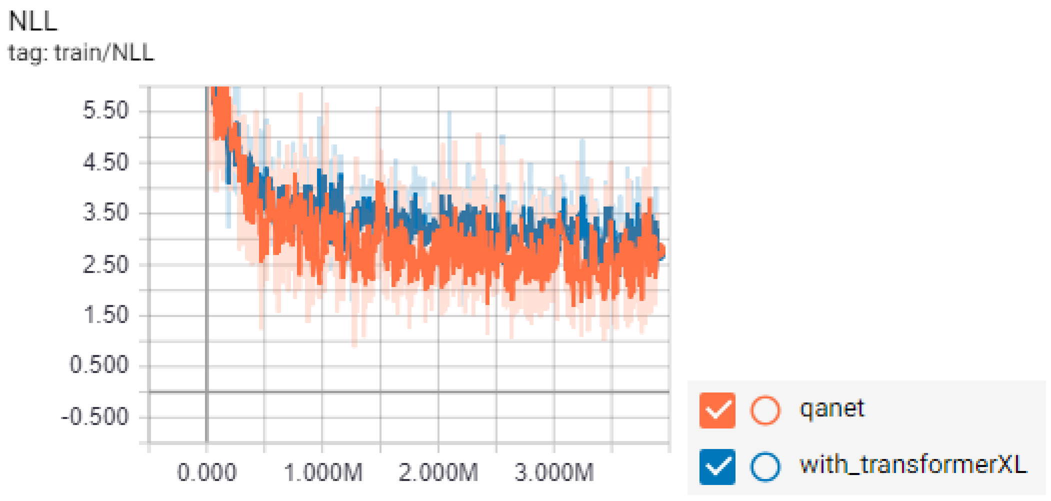

Figure 6: Training Loss

- The hyperparameters are tuned for the dev set and are not performing well on the test set

5.2 Analysis for QANet with TransformerXL extension

This extension to the QANet architecture underperformed more than expected. Some of the reasons for the underperformance are as follows

- The training was only run once due to time constraints which did not permit hyperparameter tuning
- Some debugging is needed to ensure all layers are contributing to expectation
- Some hyperparameter values were lowered (as seen in Table 1) to overcome memory and time constraints
- Decreased number of encoder blocks - the number was decreased from 7 to 1 and could have been a major contributor to the lower F1 scores. This was done to avoid memory issues

Though the training and dev NLL matched the profile seen for QANet the values for NLL did not go down to the extent seen with QANet due to reasons elaborated above.

6 Conclusion

Two models were implemented - a QANet implementation as well as a QANet implementation with TransformerXL extensions. Code was written in pytorch and the models were evaluated using the SQuAD 2.0 dataset.

Further tuning of the models can be done and requires higher GPU memory than was used in this paper. Further, other techniques like data augmentation, use of pre-trained contextual word embeddings can be incorporated to boost performance.

References

[1] Adams Wei Yu, David Dohan, Minh-Thang Luong, Rui Zhao, Kai Chen, Mohammad Norouzi, and Quoc V Le. Qanet: Combining local convolution with global self-attention for reading comprehension. *arXiv preprint arXiv:1804.09541*, 2018.

[2] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, pages 5998–6008, 2017.

[3] Min Joon Seo, Aniruddha Kembhavi, Ali Farhadi, and Hannaneh Hajishirzi. Bidirectional attention flow for machine comprehension. *CoRR*, abs/1611.01603, 2016. URL `http://arxiv.org/abs/1611.01603`.

[4] Zihang Dai, Zhilin Yang, Yiming Yang, Jaime G. Carbonell, Quoc V. Le, and Ruslan Salakhutdinov. Transformer-xl: Attentive language models beyond a fixed-length context. *CoRR*, abs/1901.02860, 2019. URL `http://arxiv.org/abs/1901.02860`.

[5] Sepp Hochreiter, Yoshua Bengio, Paolo Frasconi, Jürgen Schmidhuber, et al. Gradient flow in recurrent nets: the difficulty of learning long-term dependencies, 2001.

[6] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256, 2010.

[7] Bryan McCann, Nitish Shirish Keskar, Caiming Xiong, and Richard Socher. The natural language decathlon: Multitask learning as question answering. *CoRR*, abs/1806.08730, 2018. URL `http://arxiv.org/abs/1806.08730`.

[8] Matthew E Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. Deep contextualized word representations. *arXiv preprint arXiv:1802.05365*, 2018.

[9] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[10] Rupesh Kumar Srivastava, Klaus Greff, and Jürgen Schmidhuber. Highway networks. *arXiv preprint arXiv:1505.00387*, 2015.

[11] Lei Jimmy Ba, Ryan Kiros, and Geoffrey E. Hinton. Layer normalization. *CoRR*, abs/1607.06450, 2016. URL `http://arxiv.org/abs/1607.06450`.

[12] Omer Levy, Minjoon Seo, Eunsol Choi, and Luke Zettlemoyer. Zero-shot relation extraction via reading comprehension. *arXiv preprint arXiv:1706.04115*, 2017.

[13] Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. Squad: 100,000+ questions for machine comprehension of text. *arXiv preprint arXiv:1606.05250*, 2016.

[14] Pranav Rajpurkar, Robin Jia, and Percy Liang. Know what you don't know: Unanswerable questions for squad. *arXiv preprint arXiv:1806.03822*, 2018.

[15] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014. URL `http://arxiv.org/abs/1412.6980`.

[16] Gao Huang, Yu Sun, Zhuang Liu, Daniel Sedra, and Kilian Q Weinberger. Deep networks with stochastic depth. In *European conference on computer vision*, pages 646–661. Springer, 2016.