|  |  |
| --- | --- |
| Gerb-BMSTU_01 | Министерство науки и высшего образования Российской Федерации  Федеральное государственное бюджетное образовательное учреждение  высшего образования  «Московский государственный технический университет  имени Н.Э. Баумана  (национальный исследовательский университет)»  (МГТУ им. Н.Э. Баумана) |

|  |  |
| --- | --- |
| ФАКУЛЬТЕТ | Информатика, искусственный интеллект и системы управления (ИУ) |

|  |  |
| --- | --- |
| КАФЕДРА | Системы обработки информации и управления |

|  |  |
| --- | --- |
| дисциплина | Методы машинного обучения |

|  |
| --- |
| отчет по домашнему заданию № 1 |

|  |
| --- |
| Разбор статьи «Conformer: сверточно-дополненный трансформер для распознавания речи» |
| *название домашнего задания* |

|  |  |
| --- | --- |
| Группа | ИУ5-24М |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Студент | 01.06.2022 |  |  |  | Матиенко А. П. |
|  | *дата выполнения работы* |  | *подпись* |  | *фамилия, и.о.* |

|  |  |  |  |
| --- | --- | --- | --- |
| Преподаватель |  |  | Гапанюк Ю. E. |
|  | *подпись* |  | *фамилия, и.о.* |

Москва, 2022 г.

# Постановка задачи

Изучить модель, сочетающую в себе лучшие качества трансформеров и сверток, для распознавания речи. Ссылки на оригинальные статьи: <https://arxiv.org/pdf/2005.08100v1.pdf>, <https://arxiv.org/pdf/1906.02762.pdf>.

# Теоретическая часть

Недавно трансформер и сверточная нейронная сеть (CNN) показали многообещающие результаты в автоматическом распознавании речи (ASR), превосходящие рекуррентные нейронные сети.

*Трансформаторные модели хороши в захвате глобальных взаимодействий на основе контента, в то время как CNNS используют эффективно местные особенности.*

В этой работе изучается, как объединить сверточные нейронные сети сети и трансформаторы для моделирования как локальных, так и глобальных зависимостей аудиопоследовательности эффективным по параметрам способом.

Предлагается сверточно-дополненный трансформатор для распознавания речи, названный Конформером.

Конформер значительно превосходит Трансформатор и CNN на широко используемом эталоне LibriSpeech, модель достигает WER 2,1% / 4,3% без использования языковой модели.

В последнее время трансформеры, основанные на самосознании, получили широкое распространение для моделирования последовательностей благодаря своей способности улавливать взаимодействия на больших расстояниях.

В качестве альтернативы были также использованы свертки, которые фиксируют локальный контекст постепенно через локальное восприимчивое поле слой за слоем.

Однако трансформеры и свертки имеют свои ограничения. В то время как трансформеры хороши в моделировании в долгосрочном глобальном контексте они менее способны извлекать мелкозернистые локальные характерные узоры.

С другой стороны, сверточные нейронные сети (CNNS) используют локальную информацию. Одним из ограничений использования локальной связи является что нужно еще много слоев или параметров для захвата глобальной информации.

В этой работе изучается, как органично сочетать свертки с self-attention в моделях автоматического распознавания речи (АРР). Выдвигается гипотеза что как глобальные, так и локальные взаимодействия важны для того, чтобы параметры были эффективны. Для достижения этой цели предлагается новое

сочетание self-attention и свертки, позволит достичь наилучших результатов.

Оба мира – self-attention (самовнимание) учится глобальному взаимодействию, в то время как свертка эффективно улавливает относительное смещение на основе локальной корреляции. Вводится новое сочетание self-attention и свертки.

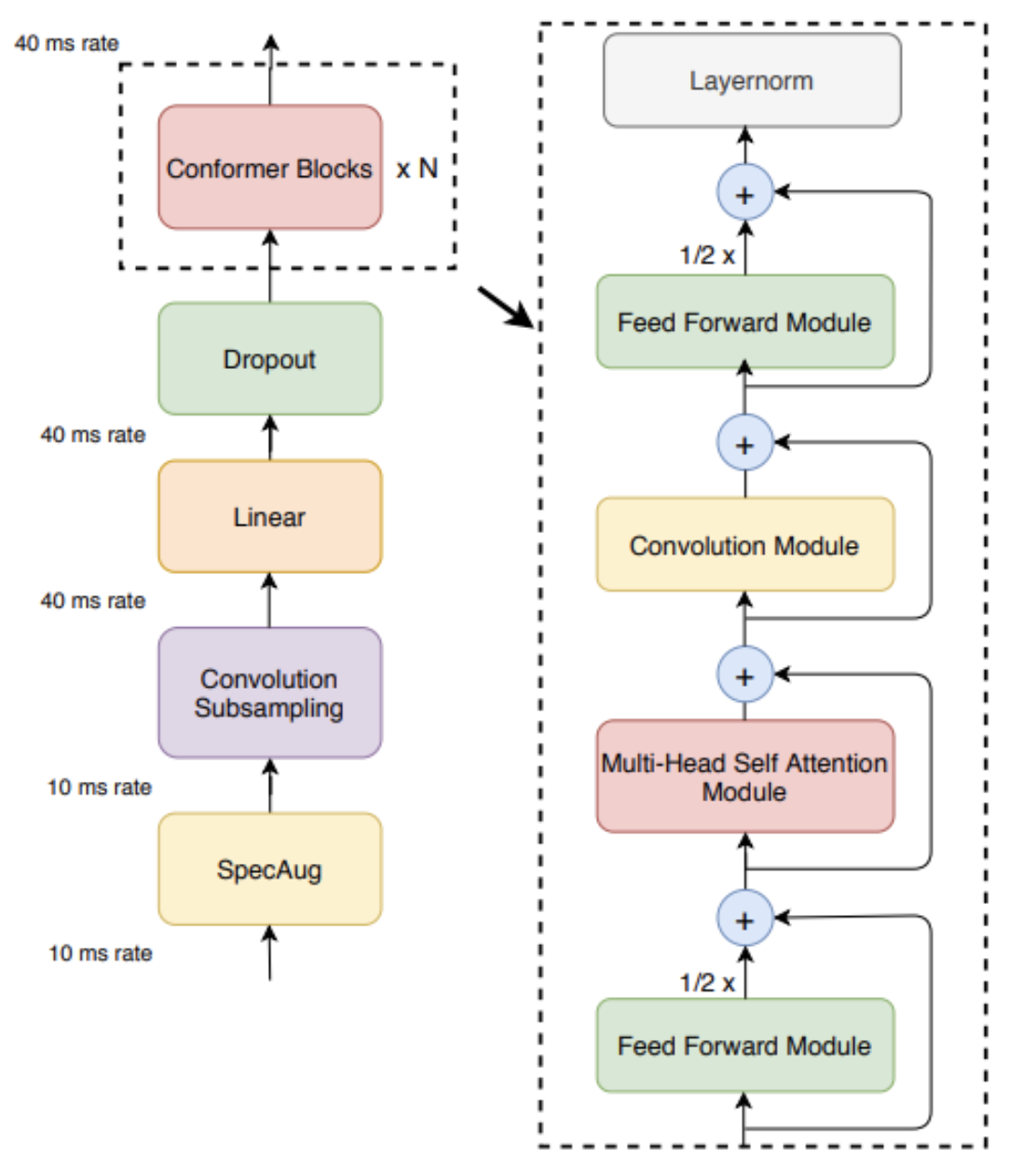


Рисунок 1 – Архитектура Conformer

Аудиокодер сначала обрабатывает входные данные с помощью свертки, а рядом conformer blocks, как показано на рисунке 1. Отличительной особенностью модели является использование conformer blocks вместо transformer blocks.

Conformer block состоит из четырех модулей, уложенных друг над другом:

* feed forward module;
* multihead self-attention module;
* convolution module;
* feed forward module.

## Multihead self-attention module

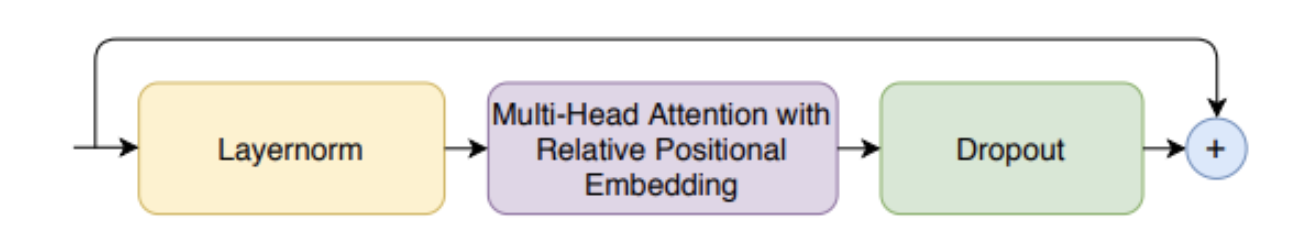


Рисунок 2 – Архитектура multihead self-attention module

Multihead self-attention (рисунок 2) модуль используется для интеграции важной части из трансформеров. Относительное позиционное кодирование, использующееся в self-attention, позволяет обобщить лучше на разную длину входного сигнала, и результирующий кодер будет более устойчив к дисперсии длины высказывания. Используется layernorm с dropout, который помогает обучению, регуляризируя более глубокие слои.

## Convolution module (Модуль свертки)

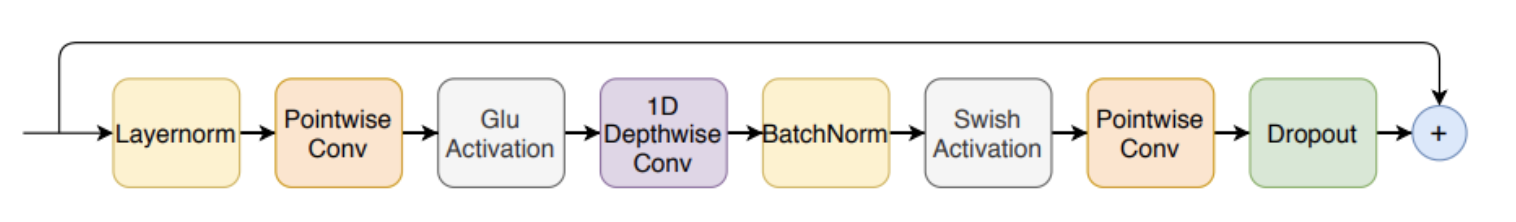


Рисунок 3 – Архитектура модуля свертки

Модуль свертки (рисунок 3) представляет собой:

* layernorm;
* pointwise conv;
* glu (gated linear unit) activation;
* 1d depthwise conv;
* batchnorm - для помощи в обучении глубоким слоям;
* swish activation;
* pointwise activation;
* dropout.

## Feed forward module

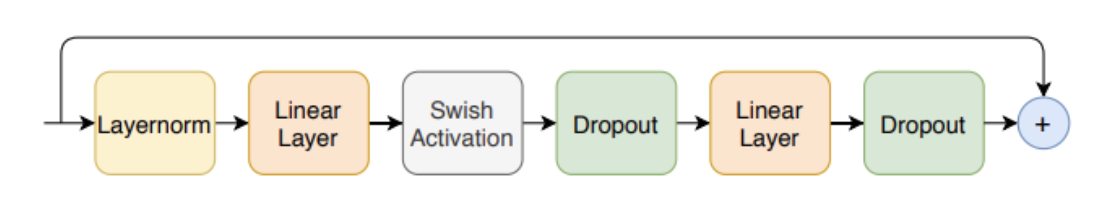


Рисунок 4 – Архитектура feed forward module

Feed forward модуль (рисунок 4) состоит из:

* layernorm;
* linear layer;
* swish activation;
* dropout;
* linear layer;
* dropout;
* residual connection.

## Conformer block

Конформер блок состоит из двух feed forward модулей, между которыми расположены multihead self-attention модуль и convolution модуль (модуль свертки), то есть архитектура трансформера и свертки.

Эта сэндвич-структура вдохновлена Macaron-Net[17], которая предлагает заменить исходный слой обратной связи (feedforward module) в трансформере на два полушаговых feedforward modules, один до слоя self-attention и один после него. Как и в Macron-Net, используются полушаговые остаточные веса в FFM. За вторым модулем прямой передачи следует последний слой - нормализация. Математически модуль conformer представлен на рисунке 5.

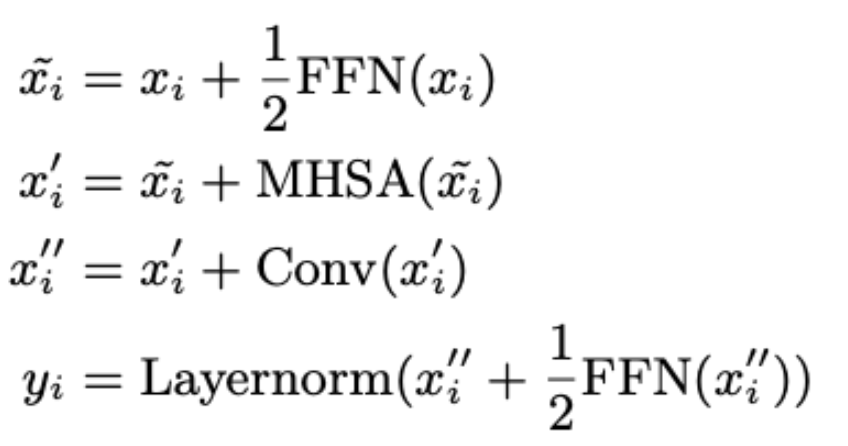


Рисунок 5 – Математическое представление conformer

* *FFN - feedforward module;*
* *MHSA - multihead self-attention;*
* *Conv - модуль свертки;*
* *Layernorm - слой нормализации.*

# Практическая часть

## Conformer Block

На листинге 1 представлена реализация conformer block.

|  |
| --- |
| class Swish(tf.keras.layers.Layer):  def \_\_init\_\_(self, \*\*kwargs):  super(Swish, self).\_\_init\_\_(\*\*kwargs)  def call(self, inputs):  return inputs \* tf.sigmoid(inputs)  class GLU(tf.keras.layers.Layer):  def \_\_init\_\_(self, dim, \*\*kwargs):  super(GLU, self).\_\_init\_\_(\*\*kwargs)  self.dim = dim  def call(self, inputs):  out, gate = tf.split(inputs, 2, axis=self.dim)  return out \* tf.sigmoid(gate)  class DepthwiseLayer(tf.keras.layers.Layer):  def \_\_init\_\_(self, chan\_in, chan\_out, kernel\_size, padding, \*\*kwargs):  super(DepthwiseLayer, self).\_\_init\_\_(\*\*kwargs)  self.padding = padding  self.chan\_in = chan\_in  self.conv = tf.keras.layers.Conv1D(chan\_out, 1, groups=chan\_in)  def call(self, inputs):  inputs = tf.reshape(inputs, [-1])  padded = tf.zeros(  [self.chan\_in \* self.chan\_in] - tf.shape(inputs), dtype=inputs.dtype  )  inputs = tf.concat([inputs, padded], 0)  inputs = tf.reshape(inputs, [-1, self.chan\_in, self.chan\_in])  return self.conv(inputs)  class Scale(tf.keras.layers.Layer):  def \_\_init\_\_(self, scale, fn, \*\*kwargs):  super(Scale, self).\_\_init\_\_(\*\*kwargs)  self.scale = scale  self.fn = fn  def call(self, inputs, \*\*kwargs):  return self.fn(inputs, \*\*kwargs) \* self.scale  class PreNorm(tf.keras.layers.Layer):  def \_\_init\_\_(self, dim, fn, \*\*kwargs):  super(PreNorm, self).\_\_init\_\_(\*\*kwargs)  self.norm = tf.keras.layers.LayerNormalization(axis=-1)  self.fn = fn  def call(self, inputs, \*\*kwargs):  inputs = self.norm(inputs)  return self.fn(inputs, \*\*kwargs)  class FeedForward(tf.keras.layers.Layer):  def \_\_init\_\_(self, dim, mult=4, dropout=0.0, \*\*kwargs):  super(FeedForward, self).\_\_init\_\_(\*\*kwargs)  self.net = tf.keras.Sequential(  [  tf.keras.layers.Dense(dim \* mult, activation=Swish()),  tf.keras.layers.Dropout(dropout),  tf.keras.layers.Dense(dim, input\_dim=dim \* mult),  tf.keras.layers.Dropout(dropout),  ]  )  def call(self, inputs):  return self.net(inputs)  class BatchNorm(tf.keras.layers.Layer):  def \_\_init\_\_(self, causal, \*\*kwargs):  super(BatchNorm, self).\_\_init\_\_(\*\*kwargs)  self.causal = causal  def call(self, inputs):  if not self.causal:  return tf.keras.layers.BatchNormalization(axis=-1)(inputs)  return tf.identity(inputs)  class ConformerConvModule(tf.keras.layers.Layer):  def \_\_init\_\_(  self,  dim,  causal=False,  expansion\_factor=2,  kernel\_size=31,  dropout=0.0,  \*\*kwargs  ):  super(ConformerConvModule, self).\_\_init\_\_(\*\*kwargs)  inner\_dim = dim \* expansion\_factor  if not causal:  padding = (kernel\_size // 2, kernel\_size // 2 - (kernel\_size + 1) % 2)  else:  padding = (kernel\_size - 1, 0)  self.net = tf.keras.Sequential(  [  tf.keras.layers.LayerNormalization(axis=-1),  Rearrange("b n c -> b c n"),  tf.keras.layers.Conv1D(filters=inner\_dim \* 2, kernel\_size=1),  GLU(dim=1),  DepthwiseLayer(  inner\_dim, inner\_dim, kernel\_size=kernel\_size, padding=padding  ),  BatchNorm(causal=causal),  Swish(),  tf.keras.layers.Conv1D(filters=dim, kernel\_size=1),  tf.keras.layers.Dropout(dropout),  ]  )  def call(self, inputs):  return self.net(inputs)  class ConformerBlock(tf.keras.layers.Layer):  def \_\_init\_\_(  self,  dim,  dim\_head=64,  heads=8,  ff\_mult=4,  conv\_expansion\_factor=2,  conv\_kernel\_size=31,  attn\_dropout=0.0,  ff\_dropout=0.0,  conv\_dropout=0.0,  \*\*kwargs  ):  super(ConformerBlock, self).\_\_init\_\_(\*\*kwargs)  self.ff1 = FeedForward(dim=dim, mult=ff\_mult, dropout=ff\_dropout)  self.attn = Attention(  dim=dim, dim\_head=dim\_head, heads=heads, dropout=attn\_dropout  )  self.conv = ConformerConvModule(  dim=dim,  causal=False,  expansion\_factor=conv\_expansion\_factor,  kernel\_size=conv\_kernel\_size,  dropout=conv\_dropout,  )  self.ff2 = FeedForward(dim=dim, mult=ff\_mult, dropout=ff\_dropout)  self.attn = PreNorm(dim, self.attn)  self.ff1 = Scale(0.5, PreNorm(dim, self.ff1))  self.ff2 = Scale(0.5, PreNorm(dim, self.ff2))  self.post\_norm = tf.keras.layers.LayerNormalization(axis=-1)  def call(self, inputs, mask=None):  inputs = self.ff1(inputs) + inputs  inputs = self.attn(inputs, mask=mask) + inputs  inputs = self.conv(inputs) + inputs  inputs = self.ff2(inputs) + inputs  inputs = self.post\_norm(inputs)  return inputs |

Листинг 1 – Реализация Conformer Block

## Self-attention

На листинге 2 представлена реализация self-attention.

|  |
| --- |
| class Attention(tf.keras.layers.Layer):  def \_\_init\_\_(  self, dim, heads=8, dim\_head=64, dropout=0.0, max\_pos\_emb=512, \*\*kwargs  ):  super(Attention, self).\_\_init\_\_(\*\*kwargs)  inner\_dim = dim\_head \* heads  self.heads = heads  self.scale = dim\_head \*\* -0.5  self.to\_q = tf.keras.layers.Dense(inner\_dim, use\_bias=False)  self.to\_kv = tf.keras.layers.Dense(inner\_dim \* 2, use\_bias=False)  self.to\_out = tf.keras.layers.Dense(dim)  self.max\_pos\_emb = max\_pos\_emb  self.rel\_pos\_emb = tf.keras.layers.Embedding(2 \* max\_pos\_emb + 1, dim\_head)  self.dropout = tf.keras.layers.Dropout(dropout)  def call(self, inputs, context=None, mask=None, context\_mask=None):  n = inputs.shape[-2]  heads = self.heads  max\_pos\_emb = self.max\_pos\_emb  if context is None:  has\_context = False  context = inputs  else:  has\_context = True  kv = tf.split(self.to\_kv(context), num\_or\_size\_splits=2, axis=-1)  q, k, v = (self.to\_q(inputs), \*kv)  q, k, v = map(  lambda t: rearrange(t, "b n (h d) -> b h n d", h=heads), (q, k, v)  )  dots = tf.einsum("b h i d, b h j d -> b h i j", q, k) \* self.scale  seq = tf.range(n)  dist = rearrange(seq, "i -> i ()") - rearrange(seq, "j -> () j")  dist = (  tf.clip\_by\_value(  dist, clip\_value\_min=-max\_pos\_emb, clip\_value\_max=max\_pos\_emb  )  + max\_pos\_emb  )  rel\_pos\_emb = self.rel\_pos\_emb(dist)  pos\_attn = tf.einsum("b h n d, n r d -> b h n r", q, rel\_pos\_emb) \* self.scale  dots = dots + pos\_attn  if mask is not None or context\_mask is not None:  if mask is not None:  mask = tf.ones(\*inputs.shape[:2])  if not has\_context:  if context\_mask is None:  context\_mask = mask  else:  if context\_mask is None:  context\_mask = tf.ones(\*context.shape[:2])  mask\_value = -tf.experimental.numpy.finfo(dots.dtype).max  mask = rearrange(mask, "b i -> b () i ()") \* rearrange(  context\_mask, "b j -> b () () j"  )  dots = tf.where(mask, mask\_value, dots)  attn = tf.nn.softmax(dots, axis=-1)  out = tf.einsum("b h i j, b h j d -> b h i d", attn, v)  out = rearrange(out, "b h n d -> b n (h d)")  out = self.to\_out(out)  return self.dropout(out) |

Листинг 2 – Реализация self-attention

# Заключение

В этой работе представлена Conformer-архитектура, которая интегрирует компоненты от CNN и трансформеров для сквозного распознавание речи. Была изучена важность каждого из компонент и продемонстрировано, что включение свертки имеет решающее значение для производительности conformer модели.

Модель демонстрирует лучшую точность при меньшем количестве параметров, чем предыдущая работа над набором данных LibriSpeech и достижение новой современной производительности на уровне 1,9% / 3,9% для теста / testother.

# Список использованных источников

[1] C.-C. Chiu, T. N. Sainath, Y. Wu, R. Prabhavalkar, P. Nguyen,

Z. Chen, A. Kannan, R. J. Weiss, K. Rao, E. Gonina et al., «Stateof-the-art speech recognition with sequence-to-sequence models» in 2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP). IEEE, 2018, pp. 4774–4778.

[2] K. Rao, H. Sak, and R. Prabhavalkar, “Exploring architectures, data and units for streaming end-to-end speech recognition with rnn-transducer,” in 2017 IEEE Automatic Speech Recognition and Understanding Workshop (ASRU). IEEE, 2017, pp. 193–199.

[3] Y. He, T. N. Sainath, R. Prabhavalkar, I. McGraw, R. Alvarez, D. Zhao, D. Rybach, A. Kannan, Y. Wu, R. Pang, Q. Liang, D. Bhatia, Y. Shangguan, B. Li, G. Pundak, K. C. Sim, T. Bagby, S.-Y. Chang, K. Rao, and A. Gruenstein, “Streaming End-to-end Speech Recognition For Mobile Devices,” in Proc. ICASSP, 2019.

[4] T. N. Sainath, Y. He, B. Li, A. Narayanan, R. Pang, A. Bruguier, S.-y. Chang, W. Li, R. Alvarez, Z. Chen, and et al., “A streaming on-device end-to-end model surpassing server-side conventional model quality and latency,” in ICASSP, 2020.

[5] A. Graves, “Sequence transduction with recurrent neural networks,” arXiv preprint arXiv:1211.3711, 2012.

[6] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, «Attention is all you need» 2017.

[7] Q. Zhang, H. Lu, H. Sak, A. Tripathi, E. McDermott, S. Koo, and S. Kumar, “Transformer transducer: A streamable speech recognition model with transformer encoders and rnn-t loss,” in ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP). IEEE, 2020, pp. 7829–7833.

[8] J. Li, V. Lavrukhin, B. Ginsburg, R. Leary, O. Kuchaiev, J. M. Cohen, H. Nguyen, and R. T. Gadde, “Jasper: An end-to-end convolutional neural acoustic model,” arXiv preprint arXiv:1904.03288, 2019.

[9] S. Kriman, S. Beliaev, B. Ginsburg, J. Huang, O. Kuchaiev, V. Lavrukhin, R. Leary, J. Li, and Y. Zhang, “Quartznet: Deep automatic speech recognition with 1d time-channel separable convolutions,” arXiv preprint arXiv:1910.10261, 2019.

[10] W. Han, Z. Zhang, Y. Zhang, J. Yu, C.-C. Chiu, J. Qin, A. Gulati, R. Pang, and Y. Wu, «Contextnet: Improving convolutional neural networks for automatic speech recognition with global context,» arXiv preprint arXiv:2005.03191, 2020.

[11] T. N. Sainath, A.-r. Mohamed, B. Kingsbury, and B. Ramabhadran, “Deep convolutional neural networks for lvcsr,” in 2013 IEEE international conference on acoustics, speech and signal processing. IEEE, 2013, pp. 8614–8618.

[12] O. Abdel-Hamid, A.-r. Mohamed, H. Jiang, L. Deng, G. Penn, and D. Yu, “Convolutional neural networks for speech recognition,” IEEE/ACM Transactions on audio, speech, and language processing, vol. 22, no. 10, pp. 1533–1545, 2014.

[13] J. Hu, L. Shen, and G. Sun, “Squeeze-and-excitation networks,” in Proceedings of the IEEE conference on computer vision and pattern recognition, 2018, pp. 7132–7141.

[14] I. Bello, B. Zoph, A. Vaswani, J. Shlens, and Q. V. Le, “Attention augmented convolutional networks,” in Proceedings of the IEEE International Conference on Computer Vision, 2019, pp. 3286–3295.

[15] B. Yang, L. Wang, D. Wong, L. S. Chao, and Z. Tu, “Convolutional self-attention networks,” arXiv preprint arXiv:1904.03107, 2019.

[16] A. W. Yu, D. Dohan, M.-T. Luong, R. Zhao, K. Chen, M. Norouzi, and Q. V. Le, “Qanet: Combining local convolution with global self-attention for reading comprehension,” arXiv preprint arXiv:1804.09541, 2018.

[17] Y. Lu, Z. Li, D. He, Z. Sun, B. Dong, T. Qin, L. Wang, and T.-Y. Liu, «Understanding and improving transformer from a multi-particle dynamic system point of view,» arXiv preprint arXiv:1906.02762, 2019.