Московский государственный технический университет им. Н.Э. Баумана

Факультет «Информатика и системы управления»

Кафедра «Системы обработки информации и управления»



**Домашнее задание**

**ИСПОЛНИТЕЛЬ:**

Морозенков О. Н.

Группа ИУ5-23М

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

"\_\_"\_\_\_\_\_\_\_\_\_\_\_2022 г.

Москва 2022

**Целью работы** **является**: Анализ современных методов машинного обучения и их применение для решения практических задач.

# Выбранная тема: «Conformer: Convolution-дополненный трансформатор для распознавания речи»

<https://arxiv.org/pdf/2005.08100v1.pdf>

# Постановка задачи

* Изучить модель, сочетающую в себе лучшие качества трансформеров и сверток, для распознавания речи.

# Теоретическая часть

Недавно Трансформер и сверточная нейронная сеть (CNN) показали многообещающие результаты в автоматическом Распознавание речи (ASR), превосходящее 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.

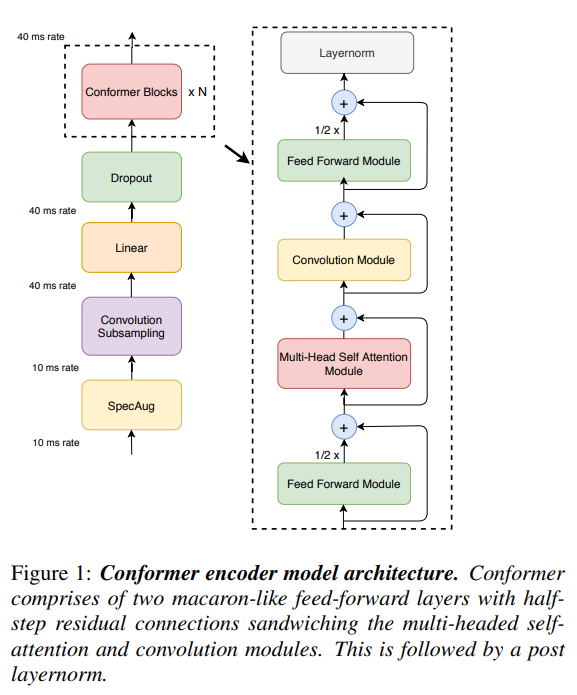
Предлагается сверточно-дополненный трансформатор для распознавания речи, названный Конформером.

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 свертки эффективно улавливают относительное смещение на основе локальной корреляции. Вводится новое сочетание self-attention и свертки



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

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

* feed forward module;
* multihead selfattention module;
* convolution module;
* feed forward module.

## Multihead self-attention module

Text, chat or text message

Description automatically generated

Multihead self-attention 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 и регуляризация более глубоких слоев.

## Convolution module (Модуль свертки)

Graphical user interface, diagram

Description automatically generated

Модуль свертки 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

Graphical user interface, diagram

Description automatically generated

Feed forward модуль состоит из:

* layernorm;
* linear layer;
* swish activation;
* dropoout;
* 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 [18],который 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 - нормальный слой. Математически это выглядит так:

Text

Description automatically generated

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

# Выводы

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.

# Практическая часть

In [5]:

**!**pip install conformer-tf

Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/

Collecting conformer-tf

Downloading conformer\_tf-0.2.0-py3-none-any.whl (11 kB)

Collecting einops~=0.3.0

Downloading einops-0.3.2-py3-none-any.whl (25 kB)

Requirement already satisfied: tensorflow>=2.5.0 in /usr/local/lib/python3.7/dist-packages (from conformer-tf) (2.8.2+zzzcolab20220527125636)

Requirement already satisfied: typing-extensions>=3.6.6 in /usr/local/lib/python3.7/dist-packages (from tensorflow>=2.5.0->conformer-tf) (4.2.0)

Requirement already satisfied: flatbuffers>=1.12 in /usr/local/lib/python3.7/dist-packages (from tensorflow>=2.5.0->conformer-tf) (2.0)

Requirement already satisfied: six>=1.12.0 in /usr/local/lib/python3.7/dist-packages (from tensorflow>=2.5.0->conformer-tf) (1.15.0)

Requirement already satisfied: tensorflow-estimator<2.9,>=2.8 in /usr/local/lib/python3.7/dist-packages (from tensorflow>=2.5.0->conformer-tf) (2.8.0)

Requirement already satisfied: keras<2.9,>=2.8.0rc0 in /usr/local/lib/python3.7/dist-packages (from tensorflow>=2.5.0->conformer-tf) (2.8.0)

Requirement already satisfied: wrapt>=1.11.0 in /usr/local/lib/python3.7/dist-packages (from tensorflow>=2.5.0->conformer-tf) (1.14.1)

Requirement already satisfied: opt-einsum>=2.3.2 in /usr/local/lib/python3.7/dist-packages (from tensorflow>=2.5.0->conformer-tf) (3.3.0)

Requirement already satisfied: protobuf<3.20,>=3.9.2 in /usr/local/lib/python3.7/dist-packages (from tensorflow>=2.5.0->conformer-tf) (3.17.3)

Requirement already satisfied: libclang>=9.0.1 in /usr/local/lib/python3.7/dist-packages (from tensorflow>=2.5.0->conformer-tf) (14.0.1)

Requirement already satisfied: setuptools in /usr/local/lib/python3.7/dist-packages (from tensorflow>=2.5.0->conformer-tf) (57.4.0)

Requirement already satisfied: gast>=0.2.1 in /usr/local/lib/python3.7/dist-packages (from tensorflow>=2.5.0->conformer-tf) (0.5.3)

Requirement already satisfied: termcolor>=1.1.0 in /usr/local/lib/python3.7/dist-packages (from tensorflow>=2.5.0->conformer-tf) (1.1.0)

Requirement already satisfied: numpy>=1.20 in /usr/local/lib/python3.7/dist-packages (from tensorflow>=2.5.0->conformer-tf) (1.21.6)

Requirement already satisfied: astunparse>=1.6.0 in /usr/local/lib/python3.7/dist-packages (from tensorflow>=2.5.0->conformer-tf) (1.6.3)

Requirement already satisfied: tensorboard<2.9,>=2.8 in /usr/local/lib/python3.7/dist-packages (from tensorflow>=2.5.0->conformer-tf) (2.8.0)

Requirement already satisfied: tensorflow-io-gcs-filesystem>=0.23.1 in /usr/local/lib/python3.7/dist-packages (from tensorflow>=2.5.0->conformer-tf) (0.26.0)

Requirement already satisfied: google-pasta>=0.1.1 in /usr/local/lib/python3.7/dist-packages (from tensorflow>=2.5.0->conformer-tf) (0.2.0)

Requirement already satisfied: grpcio<2.0,>=1.24.3 in /usr/local/lib/python3.7/dist-packages (from tensorflow>=2.5.0->conformer-tf) (1.46.3)

Requirement already satisfied: absl-py>=0.4.0 in /usr/local/lib/python3.7/dist-packages (from tensorflow>=2.5.0->conformer-tf) (1.0.0)

Requirement already satisfied: h5py>=2.9.0 in /usr/local/lib/python3.7/dist-packages (from tensorflow>=2.5.0->conformer-tf) (3.1.0)

Requirement already satisfied: keras-preprocessing>=1.1.1 in /usr/local/lib/python3.7/dist-packages (from tensorflow>=2.5.0->conformer-tf) (1.1.2)

Requirement already satisfied: wheel<1.0,>=0.23.0 in /usr/local/lib/python3.7/dist-packages (from astunparse>=1.6.0->tensorflow>=2.5.0->conformer-tf) (0.37.1)

Requirement already satisfied: cached-property in /usr/local/lib/python3.7/dist-packages (from h5py>=2.9.0->tensorflow>=2.5.0->conformer-tf) (1.5.2)

Requirement already satisfied: google-auth-oauthlib<0.5,>=0.4.1 in /usr/local/lib/python3.7/dist-packages (from tensorboard<2.9,>=2.8->tensorflow>=2.5.0->conformer-tf) (0.4.6)

Requirement already satisfied: google-auth<3,>=1.6.3 in /usr/local/lib/python3.7/dist-packages (from tensorboard<2.9,>=2.8->tensorflow>=2.5.0->conformer-tf) (1.35.0)

Requirement already satisfied: requests<3,>=2.21.0 in /usr/local/lib/python3.7/dist-packages (from tensorboard<2.9,>=2.8->tensorflow>=2.5.0->conformer-tf) (2.23.0)

Requirement already satisfied: tensorboard-data-server<0.7.0,>=0.6.0 in /usr/local/lib/python3.7/dist-packages (from tensorboard<2.9,>=2.8->tensorflow>=2.5.0->conformer-tf) (0.6.1)

Requirement already satisfied: werkzeug>=0.11.15 in /usr/local/lib/python3.7/dist-packages (from tensorboard<2.9,>=2.8->tensorflow>=2.5.0->conformer-tf) (1.0.1)

Requirement already satisfied: tensorboard-plugin-wit>=1.6.0 in /usr/local/lib/python3.7/dist-packages (from tensorboard<2.9,>=2.8->tensorflow>=2.5.0->conformer-tf) (1.8.1)

Requirement already satisfied: markdown>=2.6.8 in /usr/local/lib/python3.7/dist-packages (from tensorboard<2.9,>=2.8->tensorflow>=2.5.0->conformer-tf) (3.3.7)

Requirement already satisfied: cachetools<5.0,>=2.0.0 in /usr/local/lib/python3.7/dist-packages (from google-auth<3,>=1.6.3->tensorboard<2.9,>=2.8->tensorflow>=2.5.0->conformer-tf) (4.2.4)

Requirement already satisfied: rsa<5,>=3.1.4 in /usr/local/lib/python3.7/dist-packages (from google-auth<3,>=1.6.3->tensorboard<2.9,>=2.8->tensorflow>=2.5.0->conformer-tf) (4.8)

Requirement already satisfied: pyasn1-modules>=0.2.1 in /usr/local/lib/python3.7/dist-packages (from google-auth<3,>=1.6.3->tensorboard<2.9,>=2.8->tensorflow>=2.5.0->conformer-tf) (0.2.8)

Requirement already satisfied: requests-oauthlib>=0.7.0 in /usr/local/lib/python3.7/dist-packages (from google-auth-oauthlib<0.5,>=0.4.1->tensorboard<2.9,>=2.8->tensorflow>=2.5.0->conformer-tf) (1.3.1)

Requirement already satisfied: importlib-metadata>=4.4 in /usr/local/lib/python3.7/dist-packages (from markdown>=2.6.8->tensorboard<2.9,>=2.8->tensorflow>=2.5.0->conformer-tf) (4.11.4)

Requirement already satisfied: zipp>=0.5 in /usr/local/lib/python3.7/dist-packages (from importlib-metadata>=4.4->markdown>=2.6.8->tensorboard<2.9,>=2.8->tensorflow>=2.5.0->conformer-tf) (3.8.0)

Requirement already satisfied: pyasn1<0.5.0,>=0.4.6 in /usr/local/lib/python3.7/dist-packages (from pyasn1-modules>=0.2.1->google-auth<3,>=1.6.3->tensorboard<2.9,>=2.8->tensorflow>=2.5.0->conformer-tf) (0.4.8)

Requirement already satisfied: idna<3,>=2.5 in /usr/local/lib/python3.7/dist-packages (from requests<3,>=2.21.0->tensorboard<2.9,>=2.8->tensorflow>=2.5.0->conformer-tf) (2.10)

Requirement already satisfied: urllib3!=1.25.0,!=1.25.1,<1.26,>=1.21.1 in /usr/local/lib/python3.7/dist-packages (from requests<3,>=2.21.0->tensorboard<2.9,>=2.8->tensorflow>=2.5.0->conformer-tf) (1.24.3)

Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.7/dist-packages (from requests<3,>=2.21.0->tensorboard<2.9,>=2.8->tensorflow>=2.5.0->conformer-tf) (2022.5.18.1)

Requirement already satisfied: chardet<4,>=3.0.2 in /usr/local/lib/python3.7/dist-packages (from requests<3,>=2.21.0->tensorboard<2.9,>=2.8->tensorflow>=2.5.0->conformer-tf) (3.0.4)

Requirement already satisfied: oauthlib>=3.0.0 in /usr/local/lib/python3.7/dist-packages (from requests-oauthlib>=0.7.0->google-auth-oauthlib<0.5,>=0.4.1->tensorboard<2.9,>=2.8->tensorflow>=2.5.0->conformer-tf) (3.2.0)

Installing collected packages: einops, conformer-tf

Successfully installed conformer-tf-0.2.0 einops-0.3.2

In [6]:

**import** tensorflow **as** tf

**from** conformer\_tf **import** ConformerConvModule

**from** conformer\_tf **import** ConformerBlock

In [7]:

layer **=** ConformerConvModule(

dim**=**512,

causal**=False**, *# whether it is auto-regressive*

expansion\_factor**=**2, *# what multiple of the dimension to expand for the depthwise convolution*

kernel\_size**=**31,

dropout**=**0.0,

)

x **=** tf**.**random**.**normal([1, 1024, 512])

x **=** layer(x) **+** x *# (1, 1024, 512)*

x

Out[7]:

<tf.Tensor: shape=(1, 1024, 512), dtype=float32, numpy=

array([[[-2.1310627 , 2.2861435 , -0.69073033, ..., 0.4056328 ,

1.3223591 , 1.4631126 ],

[ 1.1534932 , 1.0052295 , 1.1302338 , ..., 0.4254702 ,

-0.13752946, -1.3404052 ],

[-1.1324432 , 0.78719646, -0.09465956, ..., 1.1944577 ,

0.8887768 , -0.05905754],

...,

[-0.22791393, -0.28866857, -0.492902 , ..., 2.3038957 ,

-2.1300359 , 1.3069866 ],

[ 0.2018835 , -0.5208937 , -0.9240337 , ..., 0.2725518 ,

0.46745616, 0.3852155 ],

[ 0.9319679 , -0.22860141, 0.30394647, ..., -0.4556215 ,

-1.2897168 , 0.21343876]]], dtype=float32)>

In [8]:

x**.**shape

Out[8]:

TensorShape([1, 1024, 512])

In [9]:

conformer\_block **=** ConformerBlock(

dim**=**512,

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,

)

x **=** tf**.**random**.**normal([1, 1024, 512])

x **=** conformer\_block(x) *# (1, 1024, 512)*

x

Out[9]:

<tf.Tensor: shape=(1, 1024, 512), dtype=float32, numpy=

array([[[-1.2838522 , 0.19151811, 0.110967 , ..., 0.07033216,

-0.37344873, 0.44585326],

[-0.32702386, 0.75032985, 0.0700893 , ..., -1.5824273 ,

-0.2518995 , -0.01304741],

[ 1.9505249 , -1.4627676 , 1.2557069 , ..., 1.3113314 ,

1.0985142 , -0.41622227],

...,

[-1.3279196 , 2.296598 , -1.3158281 , ..., -1.4573563 ,

1.4478396 , -0.573807 ],

[ 0.3415064 , 0.5932754 , -1.3160518 , ..., -0.18593895,

2.1190233 , 0.7746199 ],

[-2.68147 , 0.577585 , 0.77531147, ..., 0.44595146,

-1.9030229 , 0.27588573]]], dtype=float32)>

In [10]:

x**.**shape

Out[10]:

TensorShape([1, 1024, 512])

In [11]:

**import** einops

**import** tensorflow **as** tf

**from** einops **import** rearrange

**from** einops.layers.tensorflow **import** Rearrange

## ConformerBlock

In [12]:

**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

## Attention

In [13]:

**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)

# Список литературы

[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] Z. Wu, Z. Liu, J. Lin, Y. Lin, and S. Han, “Lite transformer with long-short range attention,” arXiv preprint arXiv:2004.11886, 2020.

[18] 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.