Белорусский государственный университет информатики и радиоэлектроники
Кафедра “Информатики и технологий программирования”

КУРСОВАЯ РАБОТА
по предмету
«Архитектура вычислительных систем»
Тема: Обзор применений разделяемой по глубине свертки.

Выполнил:
студент гр. 953506
Белоусов А. В.

Проверил:
Леченко А. В.

Минск, 2021

Содержание

Введение

В современном мире интернет-технологий нередко выносится проблема распознавания образов и глубокого обучения в целом. Суть проблемы заключается в сопоставлении некоторого рода процессов, предметов, явлений и сигналов к конкретным наборам некоторых классов путём выявления важнейших признаков и выделения их из общей массы, описав это математическим языком.

Так появились свёрточные нейронные сети – класс искусственных нейронных сетей, основанный на процессах зрительный коры человека – существуют разные классы нейронов, которые можно объединить в собственные слои, каждый из которых отвечает за реакцию на разные по сложности элементы.

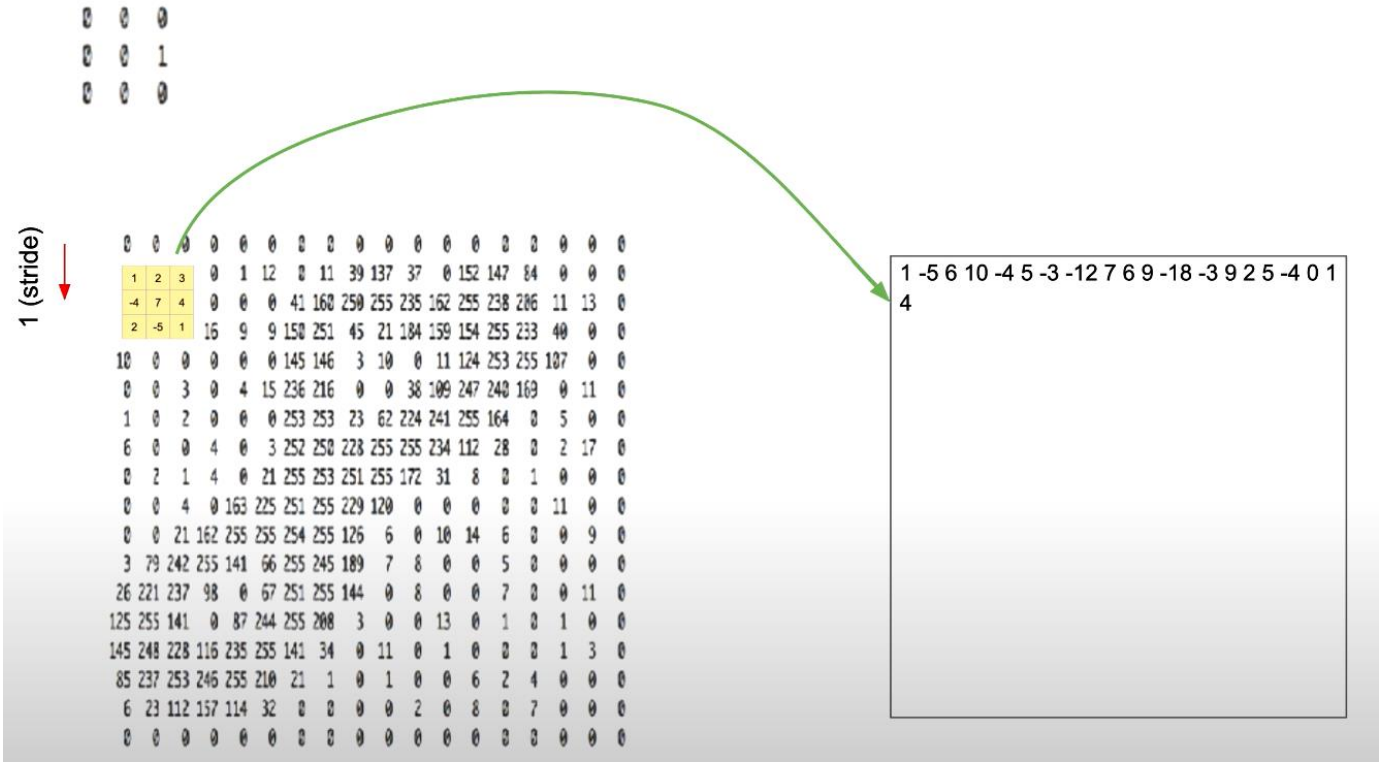
Свёрточными нейронными сетями они называются ввиду наличия операции свёртки – каждый новый элемент матрицы изображения высчитывается путём суммирования результатов умножения фрагмента текущего изображения на ядро (например 3x3 матрица весов, определяющая какой-то признак). Результатом хода свёртки является карта признаков, определяемая этим ядром. В итоге, имея большое количество таких проходов, каждый из которых формирует собственную карту, получаем большое количество таких карт. Для ускорения дальнейших вычислений за счёт уменьшения количества карт существует операция субдискретизации – из соседних нейронов выбирается максимальный, при этом считая карту признаков меньшей размерности, благодаря чему система становится более независимой от размера входных данных.

В глубоком обучении существует несколько видов свёрток, и далее мы рассмотрим разделяемую по глубине свёртку.

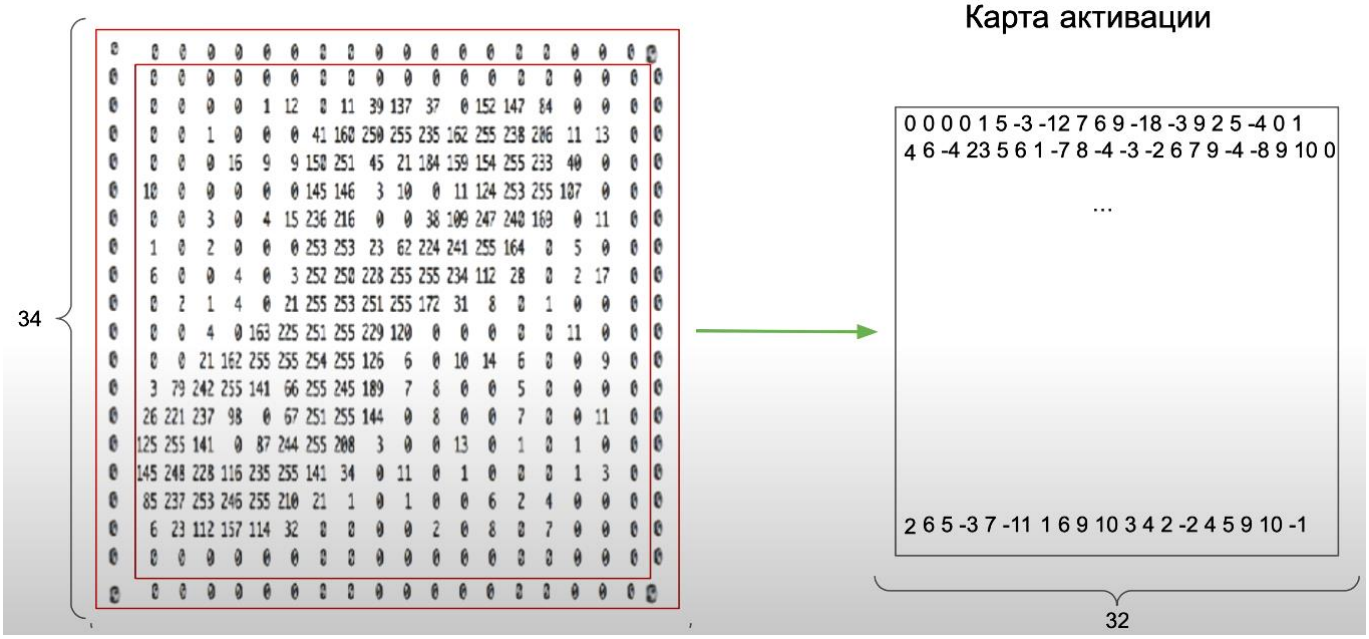
Принцип работы разделяемой по глубине свёртки

1. Свёртка

Пусть имеется фильтр (например, матрица 3x3) и входное изображение. Тогда операция свёртки – операция составления карты признаков – операция применения фильтра на каждый блок входного изображения с определённым шагом (например 1).



Итого, из изображения размера 32x32, фильтра 3x3 и шага 1 имеем выходную карту признаков размера 30x30. Также можно дополнить исходное изображение нулями таким образом, чтобы изображение имело размер 34x34 и имеем:



2. Разделяемая по глубине свёртка.

Важнейшим отличием разделяемых свёрток является то, что мы можем отделить операции взаимодействия с ядром на шаги. Таким образом, пусть y – результирующие данные, где y = conv(x, k), где x – входные данные, k – ядро, conv – операция свёртки. Пусть k вычисляется по формуле k = k1.dot(k2). В тоге мы вместо одной 2D свёртки с ядром k имеем две 1D свёртки с ядрами k1 и k2.

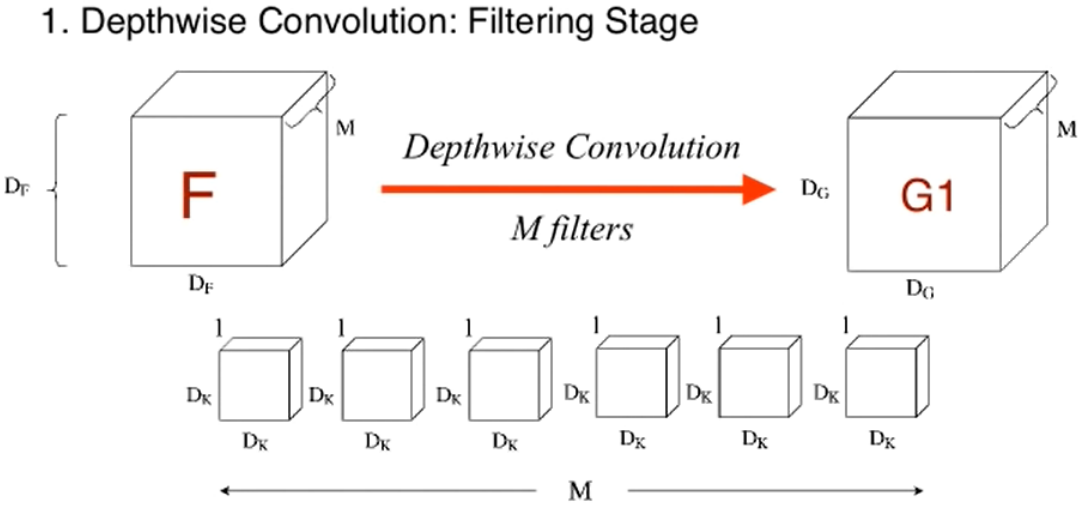
Наибольшее применение имеет разделяемая по глубине свёртка. С помощью неё мы можем выполнять пространственную свёртку, сохраняя при этом раздельные каналы, после чего выполнить свёртку по глубине. Пример:

Пусть имеется свёрточный слой размера 3x3 на 16 входных и 32 выходных каналов. Таким образом, каждый из 16 входных каналов пересекается 32 ядрами размера 3x3, итого получаем 512 карт признаков размером 16x32. Затем мы объединяем 1 карту признаков каждого входного канала, складывая их. В результате мы получаем 32 карты признаков, которые мы и ожидали получить.

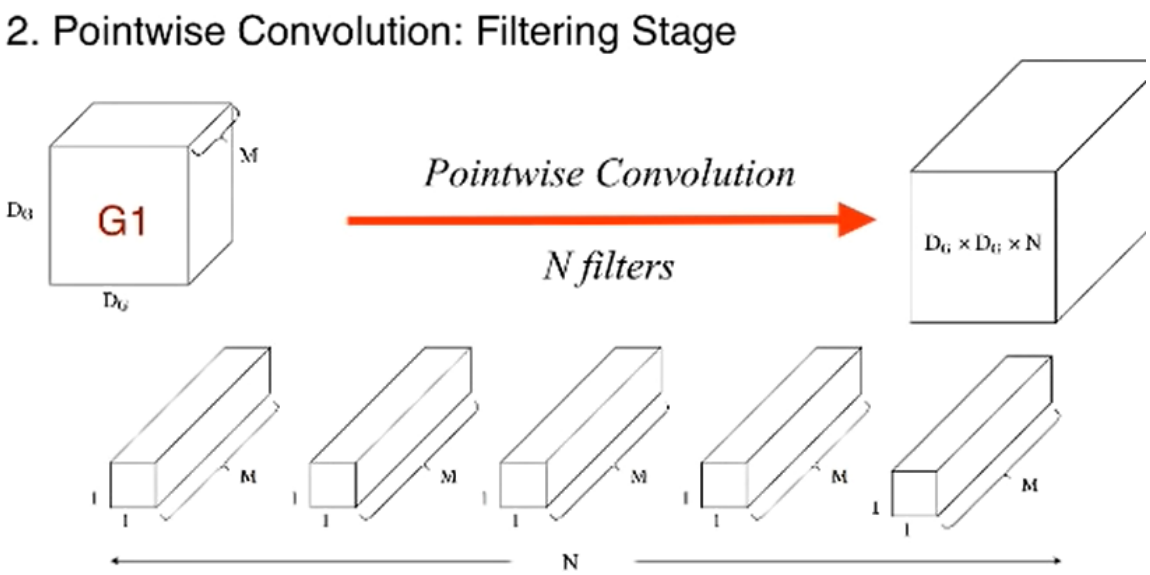
Используя разделяемую свёртку по глубине, мы проходим по каждому из 16 входных каналов с одинаковых 3x3 ядром, получая в результате 16 карт признаков. Затем мы проходимся по полученным 16 картам признаков 32 свёртками размера 1x1 и только после этого суммируем полученные результаты. В итоге имеем 16x3x3 + 16x32x1x1 = 656 весов против 16x32x3x3 = 4608 весов из примера выше. В данном примере мы рассмотрели конкретный случай, где множитель глубины равен 1 – наиболее распространённое значение.

Подводя итог, разделяемая по глубине свёртка состоит из двух свёрток:

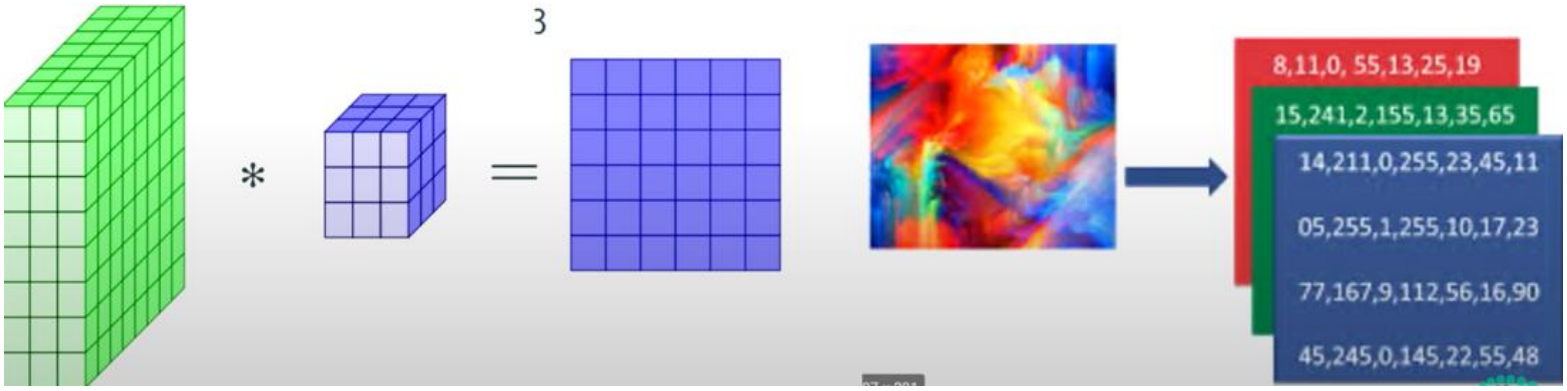
1) Свёртка в глубину a



2) Point-wise свёртка



Таким образом, 3D свёртку представляем в виде 2D свёрток по каждому каналу (в случае изображений – 3 цветовых канала RGB)



Обработка изображений нейронными сетями, использующими разделяемую по глубине свёртку

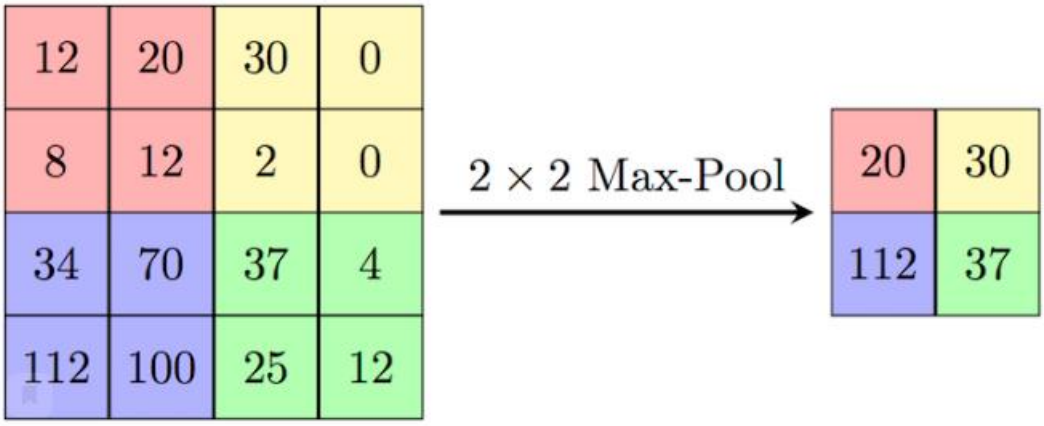
В свёрточных нейронных сетях имеется два основных слоя:

1. Свёрточный слой
2. Субдискретизирующий слой

Рассмотрим эти слои.

1. В основу свёрточного слоя входят ядра, включая при этом собственное смещение. Целью данного слоя является вычисление свёртки выходного изображения текущего слоя для каждого из ядер, суммируя при этом каждый раз смещение. После чего к выходному изображению применяется функция активации. Зачастую входной поток может состоять из n-каналов (n=1 - серая картинка, n=3 – RGB картинка и т.д.), в этом же случае ядра расширяет до такого состояния, чтобы они тоже состояли из n-каналов. В результате чего можно утверждать, что на свёрточном слое сокращается количество параметров по сравнению с обычным полносвязным слоем, но используется больше параметров, которые задаются до начала обучения нашей модели. К таким параметрам относятся: глубина – определяет количество ядер и смещение, высота и ширина ядра – зачастую берут 3x3, шаг – смещения ядра (зачастую берут 1).

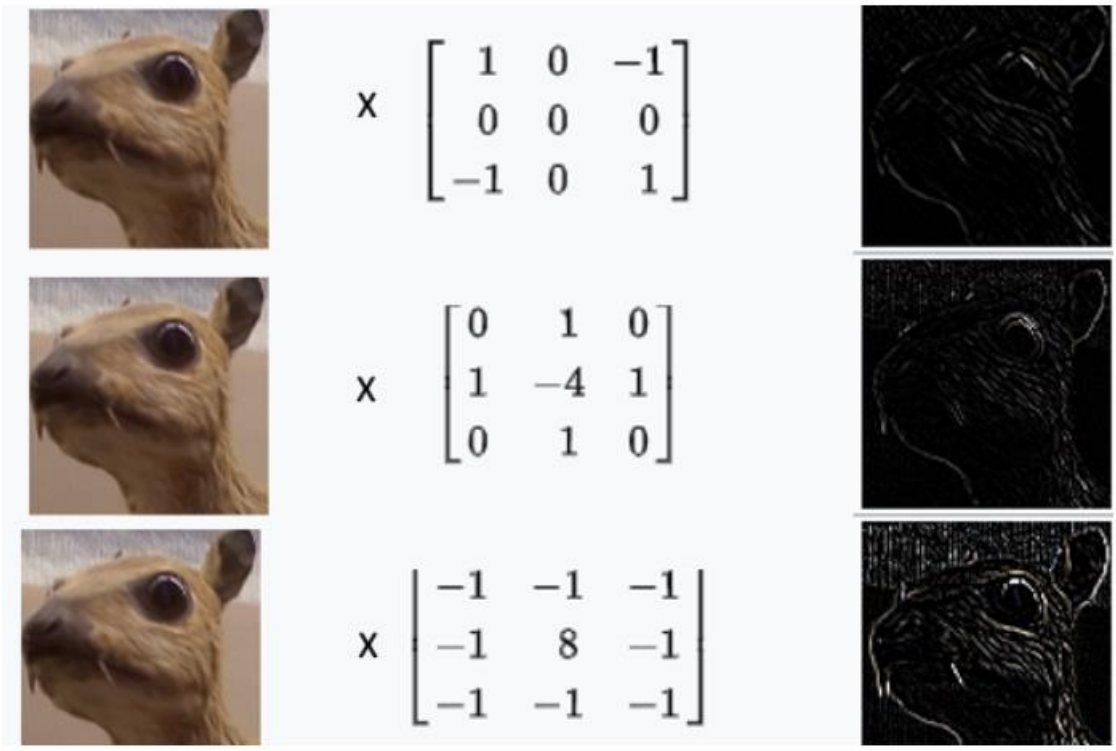
2. В основе субдискретизирующего слоя лежит идея объединения схожих фрагментов в единый фрагмент. Таким образом, имея матрицу размера 8x8, применяя на неё pool-операцию (операцию объединения 4, например – выбор максимального из 4 по значению), мы получаем матрицу меньшего размера (продолжая пример – 2x2):



Один проход через эти два слоя сокращает ширину его канала, но увеличивает его глубину(значение), т.е. выделяет какой-то определённый признак на карте признаков.

В результате неоднократного прохода этих двух слоёв и нескольких полносвязных слоев, применяя при этом функции активации после каждого свёрточного и полносвязного слоя, мы получаем выходное изображение.

Пример выполнения операции свёртки к изображению с целью выделения различных признаков на изображении в зависимости от ядра:

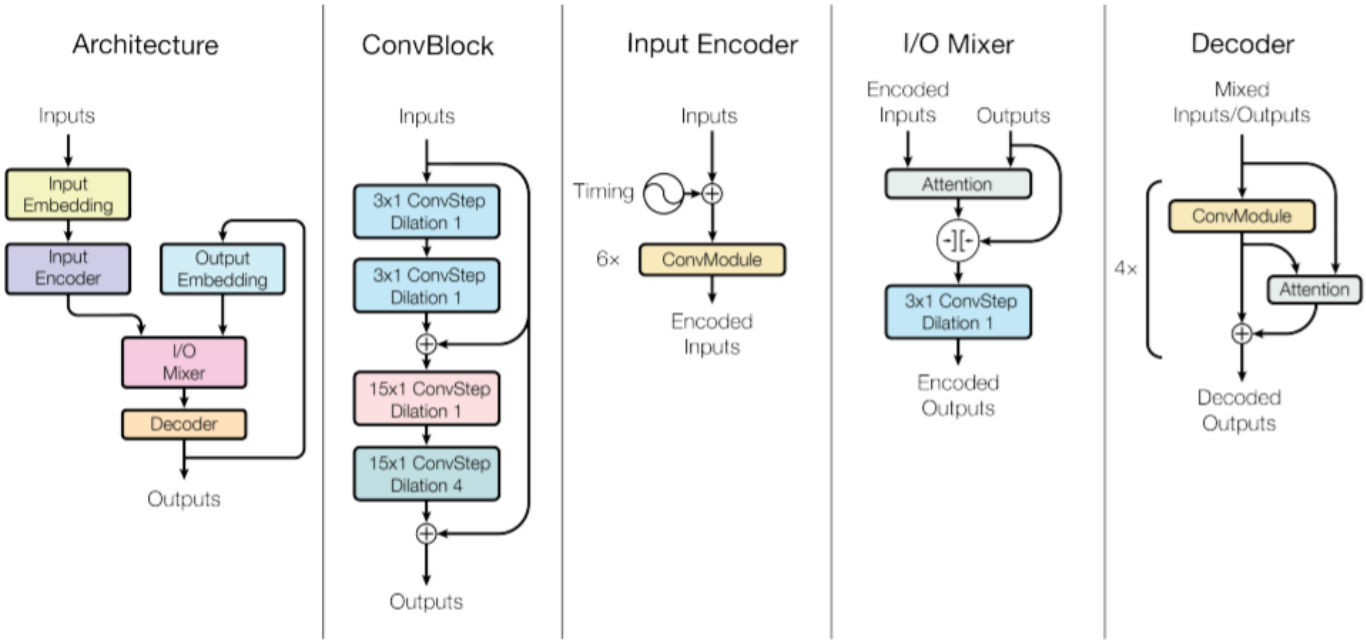


Также довольно частым явлением в снс, использующих разделяемую по глубине свёртку, является авторегрессивная модель, т.е. выходные данные модели генерируются авторегрессионным способом. В отличие от RNNS, генерация авторегрессионной последовательности зависит не только от ранее сгенерированных выходных данных, но, также от всех ранее сгенерированных выходных данных. Это понятие имеет название долгосрочных зависимостей. Установление долгосрочных зависимостей значительно повышает производительность задач RNN для NMT. Аналогично, схема генерации сверточной авторегрессии предлагает большие поля восприятия на входах и прошлых выходах, способные устанавливать эти долгосрочные зависимости.

Среди популярных свёрточных нейронных сетей, использующих разделяему по глубине свёртку можно выделить PydMobileNet(Pyramid depthwise separable convolution networks), Xception, MobileNets (mobile vision applications), Shufflenet(mobile vision), SliceNet, ByteNet, PixelCNN и другие. Рассмотрим архитектуры некоторых их них.

1)SliceNet.

Модель основана на сверточной авторегрессионной структуре. Входные и выходные значения имею одниаковую глубину объекта, кодируются двумя отдельными подсетями и объединяются перед подачей в декодер, который автоматически генерирует каждый элемент вывода. На каждом шаге авторегрессионный декодер выдает новое прогнозирование выходных данных с учетом закодированных входных данных и кодирования существующих прогнозируемых выходных данных. Кодеры и декодер основаны на ряде сверточных модулей, и attention-модуль используется для того, чтобы декодер мог получать информацию от кодера. В зависимости от реализации, меняются размеры свёрток и шаги применения фильтра. Следующая архитектура имеем наилучшие результаты в тестах на датасете Cifar-10 и Cifar-100.



Рассмотрим эти моменты подробнее.

1.1) Свёрточный модуль.

Для выполнения локальных вычислений используем модули сверток с нелинейностями ReLU и нормализацией слоев. Модуль сверток получает в качестве входных данных тензор формы и возвращает тензор той же формы. Каждый шаг в модуле состоит из трех компонентов: повторная активация входных данных с последующей разделяемой сверткой по глубине. Затем следует нормализация слоя. Нормализация слоя действует на h скрытых единиц нижнего слоя, вычисляя статистику по слоям и соответственно нормализуя. Эти нормализованные единицы затем масштабируются и сдвигаются соответственно по скалярным изученным параметрам, в результате чего конечные единицы активируются нелинейностью, где сумма берется только по последнему (глубинному) измерению. После чего сверточные результаты объединяются в модули путем их укладки и добавления остаточных соединений.

1.2) Attention модуль.

Attention механизм вычисляет сходство векторов признаков в каждой позиции и масштабирует их в соответствии с глубиной.

$$Attend(source, target) = \frac{1}{\sqrt{depth}} \cdot Softmax(target \cdot source^T) \cdot source$$

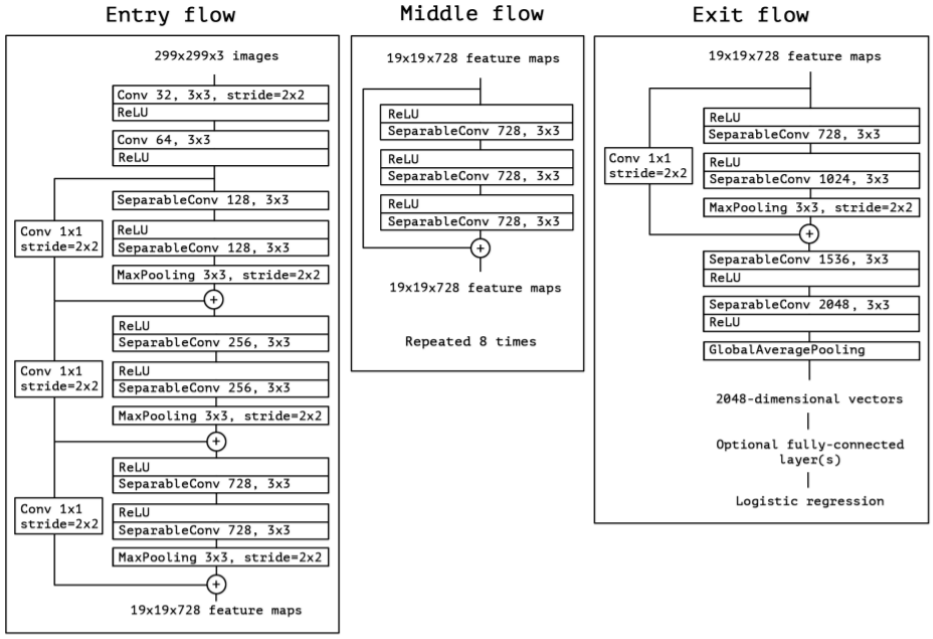
Данная модель имеет следующие результаты по истечении 250000 итераций градиентного спуска. Нетрудно заметить, что наилучшие результаты(67.27%) достигнуты при размерах ядра 3x7x15:

Dilations	Filter Size	Separability	Parameters (Non-Emb.)	Neg. Log	Accuracy
1-2-4-8	3-3-3-3	None	314 M (230 M)	-1.92	62.41
1-2-4-8	3-3-3-3	Full	196 M (112 M)	-1.83	63.87
1-1-2-4	3-7-7-7	Full	197 M (113 M)	-1.80	64.37
1-1-1-2	3-7-15-15	Full	197 M (113 M)	-1.80	64.30
1-1-1-1	3-7-15-31	Full	197 M (113 M)	-1.80	64.36
1-2-4-8	3-3-3-3	16 Groups	207 M (123 M)	-1.86	63.46
1-1-1-1	3-7-15-31	Super 2/3	253 M (141 M)	-1.78	64.71
1-1-1-1	3-7-15-31-63	Full (2048)	349 M (265 M)	-1.68	66.71
1-1-1-1	3-7-15-31	Super 2/3 (3072)	348 M (222 M)	-1.64	67.27

2) Xception

В данной реализации модель полностью основана на разделяемых по глубине слоях свертки. По сути, идея заключается в том, что межканальные корреляции и пространственные корреляции в картах объектов сверточных нейронных сетей могут быть полностью разделены. Поскольку эта гипотеза является более сильной версией гипотезы, лежащей в основе архитектуры Inception, данная архитектура называется Xception, что означает “Экстремальное начало”.

Архитектура Xception имеет 36 сверточных слоев, образующих базу для извлечения объектов сети. За сверточной базой последует слой логистической регрессии. При желании можно вставить полностью связанные слои перед слоем логистической регрессии. 36 сверточных слоев структурированы в 14 модулей, все из которых имеют линейные остаточные соединения вокруг них, за исключением первого и последнего модулей. Короче говоря, архитектура Xception представляет собой линейный стек разделяемых по глубине слоев свертки с остаточными соединениями. Это даёт возможность легко модифицировать архитектуру; это занимает всего от 30 до 40 строк кода, используя библиотеку высокого уровня, таких как Keras или TensorFlow-Slim. Реализация Xception имеет открытый исходный код.



Данная модель является модренизированной Inception V3. Сравнивая эти две модели на задачах классификации 10 и 17 тысяч классов, имеем улучшение на 10% в размере модели и времени определения класса:

Table 3. Size and training speed comparison.

	Parameter count	Steps/second
Inception V3	23,626,728	31
Xception	22,855,952	28

3) .MobileNet

Структура построена на разделяемых по глубине свертках, за исключением первого слой, который представляет собой полную свертку. Определяя сеть в таких простых терминах, мы можем легко исследовать сетевые топологии, чтобы найти хорошую сеть. Архитектура MobileNet определена в следующей таблице:

Type / Stride	Filter Shape	Input Size
Conv / s2	$3 \times 3 \times 3 \times 32$	$224 \times 224 \times 3$
Conv dw / s1	$3 \times 3 \times 32$ dw	$112 \times 112 \times 32$
Conv / s1	$1 \times 1 \times 32 \times 64$	$112 \times 112 \times 32$
Conv dw / s2	$3 \times 3 \times 64$ dw	$112 \times 112 \times 64$
Conv / s1	$1 \times 1 \times 64 \times 128$	$56 \times 56 \times 64$
Conv dw / s1	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 128$	$56 \times 56 \times 128$
Conv dw / s2	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 256$	$28 \times 28 \times 128$
Conv dw / s1	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 256$	$28 \times 28 \times 256$
Conv dw / s2	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 512$	$14 \times 14 \times 256$
$5\times$ Conv dw / s1	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
$5\times$ Conv / s1	$1 \times 1 \times 512 \times 512$	$14 \times 14 \times 512$
Conv dw / s2	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
Conv / s1	$1 \times 1 \times 512 \times 1024$	$7 \times 7 \times 512$
Conv dw / s2	$3 \times 3 \times 1024$ dw	$7 \times 7 \times 1024$
Conv / s1	$1 \times 1 \times 1024 \times 1024$	$7 \times 7 \times 1024$
Avg Pool / s1	Pool 7×7	$7 \times 7 \times 1024$
FC / s1	1024×1000	$1 \times 1 \times 1024$
Softmax / s1	Classifier	$1 \times 1 \times 1000$

За всеми слоями следует нелинейность batchnorm и ReLU, за исключением конечного полностью связанного слоя, который не имеет нелинейности и подается в слой softmax для классификации. Слой с регулярными свертками, нелинейностью batchnorm и ReLU сопоставляется с факторизованным слоем с глубинной сверткой, точечной сверткой 1×1, а также с пакетной нормой и ReLU после каждого сверточного слоя. Отбор проб вниз обрабатывается с помощью пошаговой свертки в глубинных извилинах, а также в первом слое. Окончательное среднее объединение уменьшает пространственное разрешение до 1 перед полностью подключенным слоем. Считая глубинные и точечные свертки как отдельные слои, MobileNet имеет 28 слоев.

Также важно убедиться, что эти операции могут быть эффективно

реализованы. Например, неструктурированные операции с разреженной матрицей обычно не быстрее, чем операции с плотной матрицей, до очень высокого уровня разреженности. Структура модели помещает почти все вычисления в плотные свертки 1× 1. Это может быть реализовано с помощью высокооптимизированной общей матрицы умножения

(GEMM) функции. Часто свертки реализуются GEMM, но требуют первоначального изменения порядка в памяти, называемого im2col, чтобы сопоставить его с GEMM. Свертки 1×1 не требуют такого переупорядочения в памяти и могут быть реализованы непосредственно с помощью GEMM, который является одним из наиболее оптимизированных алгоритмов численной линейной алгебры. MobileNet тратит 95 % своего вычислительного времени на свертки 1×1, которые также содержат 75 % параметров. Почти все дополнительные параметры находятся на полностью подключенном уровне.

Модели MobileNet были обучены в TensorFlow с использованием RMSProp с асинхронным градиентным спуском, аналогичным Inception V3. Однако, в отличие от обучения больших моделей, мы используем меньше методов регуляризации и увеличения данных, потому что у небольших моделей меньше проблем с переобучением. При обучении мобильных сетей мы не используем боковые головки или сглаживание надписей и дополнительно уменьшаем количество искажений изображения, ограничивая размер небольших культур, которые используются в больших начальных обучениях. Кроме того, важно было очень мало или вообще не снижать вес на глубинных фильтрах, поскольку в них и так мало параметров.

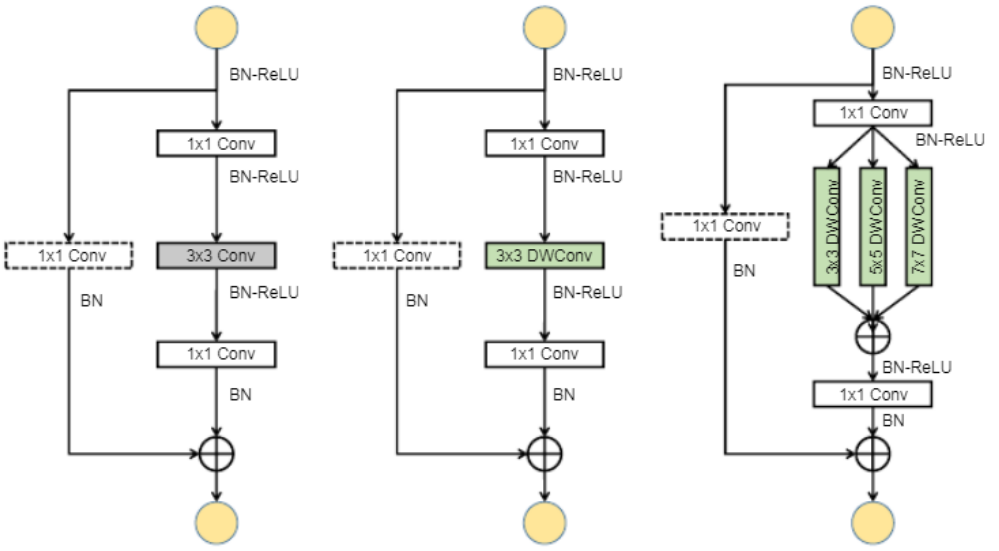
Таким образом, имеем следующие результаты:

Resolution	ImageNet Accuracy	Million Mult-Adds	Million Parameters
1.0 MobileNet-224	70.6%	569	4.2
1.0 MobileNet-192	69.1%	418	4.2
1.0 MobileNet-160	67.2%	290	4.2
1.0 MobileNet-128	64.4%	186	4.2

4) PydMobileNet.

Стоит учесть, что данная СНС используют пирамидальную разделяемую по глубине свёртку. Вместо ядра одного размера используется пирамида ядер размерности K = {k1, k2,..kn}, только после чего складываем всё воедино перед переходом к pointwise свёртке.

Основным модулем, отличающимся от схожих СНС, является Residual модуль. Свертка 1×1 изменяет каналы на коэффициент α, за которым следует основная свертка, которая может быть стандартной сверткой, или разделяемой по глубине сверткой, или пирамидальной разделяемой по глубине свёрткой, с шагом s. Наконец, другая свертка 1×1 используется для изменения количества каналов на количество выходных каналов. В основном используются три конфигурации остаточного блока. Это остаточный блок со стандартной сверткой, разделяемой по глубине и пирамидальной разделяемой по глубине свёртками:



(a) Residual block with standard Convolution (b) Residual block with Depthwise Convolution (c) Residual block with Addition Pyramid Depthwise Convolution

Основное отличие от MobileNet – наличие пирамидальной разделяемой по глубине свёртки – она использует пирамиду размера ядра K={k1, k2, ..., kn} для слоя свертки по глубине вместо одного размера ядра. Затем объединяет все выходные данные этих преобразований, прежде чем перейти к поточечному преобразованию 1×1, используя сложение. Вычислительная стоимость углубленной свертки K={k1, k2, ..., km} в случае дополнительной комбинации равна h*w*di*. $\sum_{m=0}^{M} k_m^2$ Дополнительные расходы оператора (M−1)·h*w*di. А поточечная революция 1×1 стоит h·w·di·dj. В итоге вычислительная стоимость преобразования равна h·w·di* $(M-1+\sum_{m=0}^{M} k_m^2 + d_j)$. Таким образом, соотношение вычислительных затрат стандартной свертки и преобразования PYDDW равно

$$\frac{k^2 \cdot d_j}{(M - 1 + \sum_{m=0}^{M} k_m^2 + d_j)}$$

Сравним MobileNet и PydMobileNet на CIFAR датасете (RGB картинки размером 32x32 пикселя, до 100 классов, 50000 картинок на обучение и 10000 картинок в тестировании). Моменты реализации – стандартные, опустим. Исходя из следующей таблицы, можно заметить, что PydMobileNet превосходит MobileNet: имеет меньшее количество ошибок при использовании меньшего количества параметров.

Model	Depth	#Params	FLOPs	CIFAR-10	CIFAR-100
ResNet-20*	20	0.278M	87M	7.3	-
ResNet-29-0.5	29	0.221M	29M	6.97	19.62
MobileNet-29-0.5	29	0.079M	12M	8.63	22.59
MobileNet-29-1	29	0.142M	22M	7.09	19.40
MobileNet-29-1.5	29	0.206M	32M	6.56	18.09
PydMobileNet-29-0.25	29	0.060M	10M	9.43	21.96
PydMobileNet-29-0.5	29	0.104M	18M	7.29	20.26
PydMobileNet-29-0.75	29	0.148M	26M	**6.52**	**17.95**
PydMobileNet-29-1	29	0.193M	34M	**6.00**	**17.54**
ResNet-56*	56	0.861M	277M	5.4	-
ResNet-56-0.5	56	0.435M	60M	5.76	17.60
MobileNet-56-0.5	56	0.151M	23M	6.75	18.56
MobileNet-56-1	56	0.283M	43M	6.02	17.15
MobileNet-56-1.5	56	0.416M	63M	5.29	16.58
PydMobileNet-56-0.25	56	0.109M	19M	7.38	20.41
PydMobileNet-56-0.5	56	0.200M	36M	6.19	17.36
PydMobileNet-56-0.75	56	0.292M	52M	5.55	**16.58**
PydMobileNet-56-1	56	0.382M	69M	**4.98**	**16.23**

Обработка аудио-сигналов нейронными сетями, использующими разделяемую по глубине свёртку

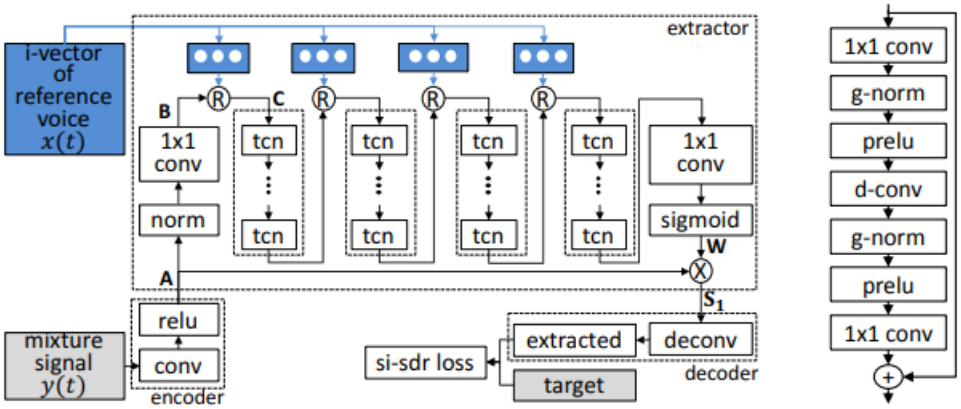
Среди основных примеров задач, решаемых с помощью нейронных сетей, использующих разделяемую по глубине свёртку, можно выделить задачи выделения вокала из композиции, определение жанра композиции и в целом выделения классов музыки, разговора и шума.

Основное отличие от нейросетей, обрабатывающих изображения, является тот факт, что при обработке аудио-сигналов мы в большинстве своем работаем с MFCC(Мел-кепстральные коэффициенты) – коэффициенты мел-частотного спектра(спектра распознаваемых человеках частот). Именно эти частоты и подаются как входные каналы(вместо RGB в изображениях).

Рассмотрим некоторые из СНС, использующих для обработки аудио-сигналов разделяемую по глубине свёртку.

1) TseNet

Основной целью данной СНС является выделение голоса спикера среди остального шума. СНС состоит из ряда других разделяемых по глубине СНС, которые улавливают зависимость речевого сигнала на большие расстояния с управляемым числом параметров. Архитектура:



(a) TseNet. (b) TCN block.

Основные элементы TseNet:

1. Кодировщик – переводит сигнал из временной области (вид представления сигналов) в своё представление, используя СНС.
2. Модуль выделения сигнала – принимает закодированный сигнал(представление), который составляет карты признаков целевого сигнала. После чего извлекается только представление необходимого нам сигнала.
3. Декодировщик – восстанавливает извлеченное представление в понятный человеку вид.

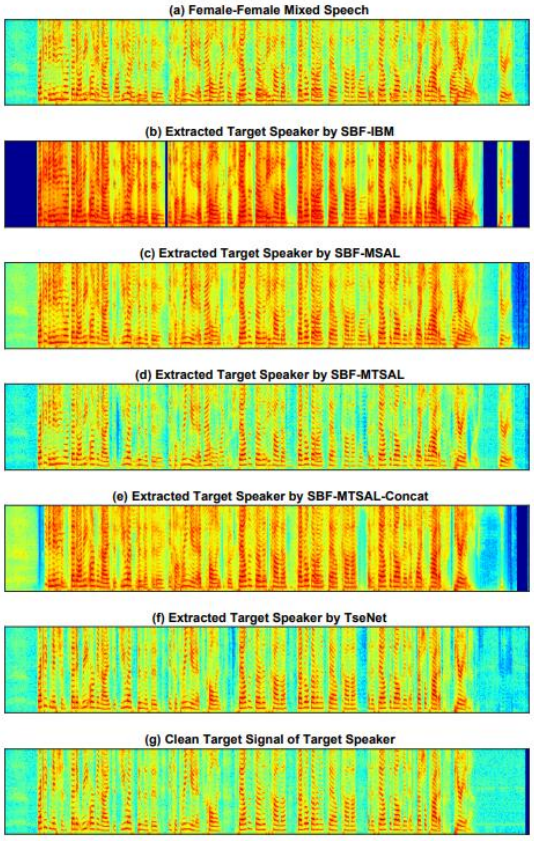
Опишем более подробно второй модуль:

В частности, кодированное представление сначала нормализуется по его среднему значению и дисперсии по размеру канала, масштабируемому с помощью обучаемых параметров смещения и усиления. Затем выполняется CNN 1×1 с N фильтрами для регулировки количества каналов для входов и остаточного пути последующих блоков TCN. В то же время вводится слой с функцией активации ReLU для приема i-вектора(целевого сигнала), а его выходы повторяются и объединяются с кодированным представлением. Закодированное представление будет содержать информацию о целевом сигнал. Нелинейный плотный слой всегда добавляется для точной настройки i-вектора динамика перед подачей его в последующие блоки TCN. Чтобы уловить зависимость речевого сигнала на большие расстояния с управляемым числом параметров, расширенные разделяемые по глубине сети свертки укладываются в несколько блоков TCN путем экспоненциального увеличения коэффициента расширения. Каждый блок TCN состоит из двух CNN 1 × 1 для определения количества входных и

выходных каналов глубинной свертки.

Первый 1×1 CNN имеет фильтры с размером ядра 1×1. Форма фильтров в глубокой свертке будет [1 × P, O], где 1 × P - размер ядра. Выходы из глубинной свертки имеют размер канала O. Второй 1×1 CNN с N фильтрами регулирует размер канала выходов глубинной свертки от O до N. Входы с N каналами блока TCN затем добавляются к его выходам с N каналами по остаточному пути, за исключением блока TCN с объединенными представлениями в качестве его входов, где кодированное представление перед объединением будет добавлено к выходам этого блока TCN. Формируем b блоков TCN воедино и повторяем в течение r раз в экстракторе. С каждым коэффициенты расширения глубинных извилин в блоках b TCN будут увеличиваться Поскольку предполагаемая маска W будет использоваться для извлечения из кодированных представлений A, размеры W должны быть такими же, как A. Наконец, извлеченное представление целевого динамика.

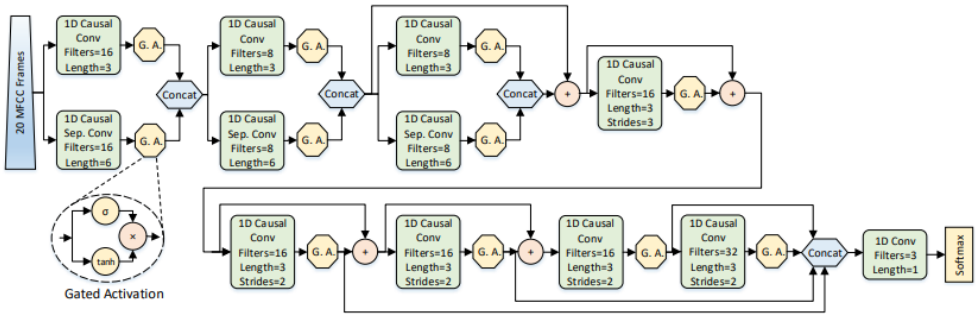
Пример иллюстрации работы TCN (выделение речи одного человека):



2) SwishNet

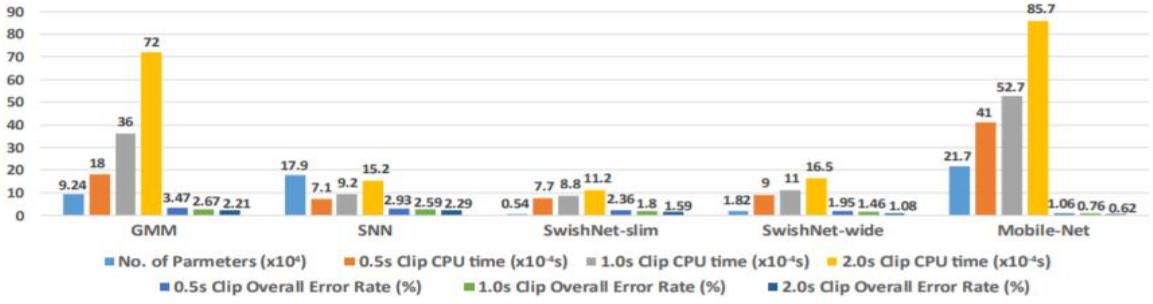
Цель – классификация музыки и разговоров.

Архитектура:



Вкратце, архитектура состоит из нескольких сверточных слоев, отсутствуют ReLU функции (вместо них используются sigmpid и tanh функции); имеются функции, позволяющие сети выбирать, какую информацию передавать с одного временного шага на следующий, что ощутимо сокращает количество карт признаков, переходящих от одного слоя к другому. CHC использует более длинные фильтры, что позволило сделать более плавный переход значений параметров с нижних на верхние слои, чтобы выделять необходимую информацию. Причины использования длинных фильтров также заключается в том, что важно отслеживать долгосрочные зависимости. Что же касается параметров разделяемых по глубине свёрточных слоёв, там используются ядра размера 6x6, выходы которых после конкатенируются.

При обучении данной модели принято использовать оптимизатор Adam. Закрытые активации, используемые в SwishNet, по-видимому, в некоторой степени вызывали проблемы с исчезающим градиентом, и обучение проходило относительно медленно. Для небольших сегментов количество обучающих выборок увеличилось, поэтому пропорционально увеличили размер пакета, чтобы сохранить количество обновлений градиента за эпоху неизменным. В данном случае тренировали SwishNet в течение 120 эпох. MobileNet обучался же очень быстро при инициализации с помощью весов ImageNet и достиг конвергенции в течение очень немногих (8-10) эпох. GMM обучался до конвергенции, а SNN обучался до тех пор, пока потери при проверке не достигли минимально допустимых. Таким образом, имеем следующие результаты(чем ниже столбец, тем лучше результаты):

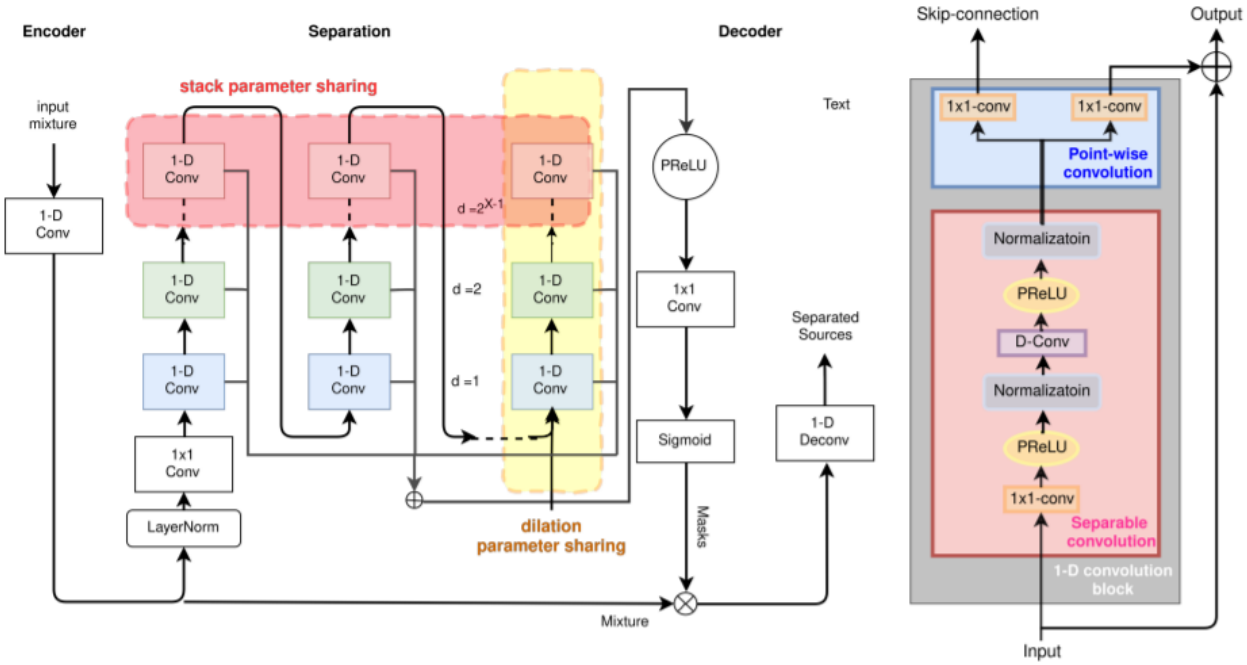


3) MiTas

Основная цель – выделение речи спикера.

Основана на TasNet, решающая основную проблему TasNet – огромное занимаемое место в памяти, путем уменьшения количества параметров (их переиспользования).

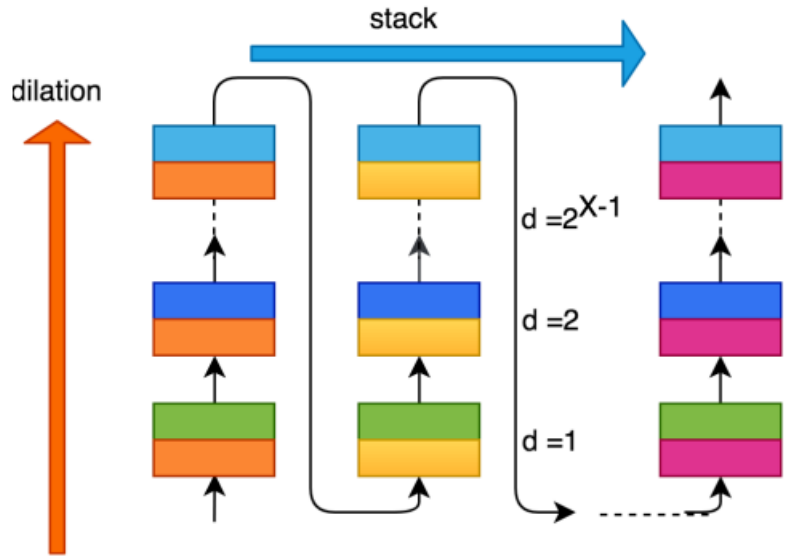
Архитектура, как рассмотренных до этого двух СНС, обрабатывающих аудио-сигналы, состоит из кодировщика, выделяющего модуля и декодировщика. Кодировщик сначала отображает сегмент смешанных сигналов в представление. Затем модель выделения сигнала оценивает маску для создания карт признаков. Наконец, декодер реконструирует разделенную исходную форму сигнала в привычный для человека вид.



(A) Conv-TasNet (B) 1-D convolution Block

Разделительный модуль составляет основную часть общего размера модели. Он составляет почти 98 % от общего числа параметров. Блок свертки в расширенной во времени сверточной сети состоит из разделяемой свертки и операции point-wise свертки.

Собственно сжатие модели за счёт параметров заключается в том, что мы используем одинаковые параметры на разных уровнях. Таким образом, что все базовые компоненты с разными расширениями в одном и том же месте имеют одинаковые веса. Реализовано это благодаря следующей логике согласно изображению: все стеки имеют одинаковые веса, т.е. все базовые элементы с одинаковыми расширениями имеют одинаковые веса. В итоге имеем, что разделяемые свертки разделяются через расширения, в то время как point-wise разделяются через стеки.



Таким образом, данная модель, обученная на датасетах WSJ0-2mix (30 часов для обучения и 10 проверки), имеет следующие результаты(трёхкратное улучшение в размере и вычислительных операциях по сравнению с TasNet):

Ablation Model* (shared dimension)	**separable conv.** stack	dil.	point-wise conv. stack	dil	**Size**	**C.P.**	**SI-SNRi (dB)**	**SDRi (dB)**
TasNet (Base-Model)	-	-	-	-	**8.9 M**	100%	**16.15**	**16.41**
MiTAS_ns **	-	-	V	-	5.6M	64%	15.61	15.88
MiTAS_sn	V	-	-	-	5.5M	62.7%	15.59	15.86
MiTAS_nd	-	-	-	V	5.1M	58.7%	15.28	15.55
MiTAS_dn	-	V	-	-	4.9M	56.51%	15.08	15.35
MiTAS_na	-	-	V	V	4.7M	53.5%	15.03	15.31
MiTAS_ss	**V**	**-**	**V**	**-**	**2.3M**	**26.7%**	**15.01**	**15.42**
—- with H=1024	**V**	**-**	**V**	**-**	**4.5M**	**52.2%**	**15.30**	**15.56**
MiTAS_ss (6 stacks)	**V**	**-**	**V**	**-**	**2.3M**	**26.7%**	**14.84**	**15.15**
Simplified-Base-Model 1	only 1 stack				2.3M	26.7%	13.10	13.29
Simplified-Base-Model 2	only 1 block in each stack				1.3M	14.6%	8.53	8.76
Conv-TasNet (Base-Model)	-	-	-	-	**5.1M**	**100%**	**14.79**	**15.05**
Conv-MiTAS_sn	V	-	-	-	3.9M	77%	14.28	14.55
Conv-MiTAS_nd	-	-	V	-	2.9M	58%	14.28	14.55
Conv-MiTAS_ss	**V**	**-**	**V**	**-**	**1.8M**	**36%**	**14.22**	**14.49**
—- with H=1024	**V**	**-**	**V**	**-**	**3.4M**	**68%**	**14.93**	**15.21**
Conv-MiTAS_ss (5 stacks)	**V**	**-**	**V**	**-**	**1.8M**	**36%**	**14.12**	**14.42**
Simplified-Base-Model 1	only 1 stack				1.8M	36%	12.50	12.76
Simplified-Base-Model 2	only 1 block in each stack				0.8M	16%	8.01	8.27

* Only partial results are listed in the table. The whole results can be found in Fig 3

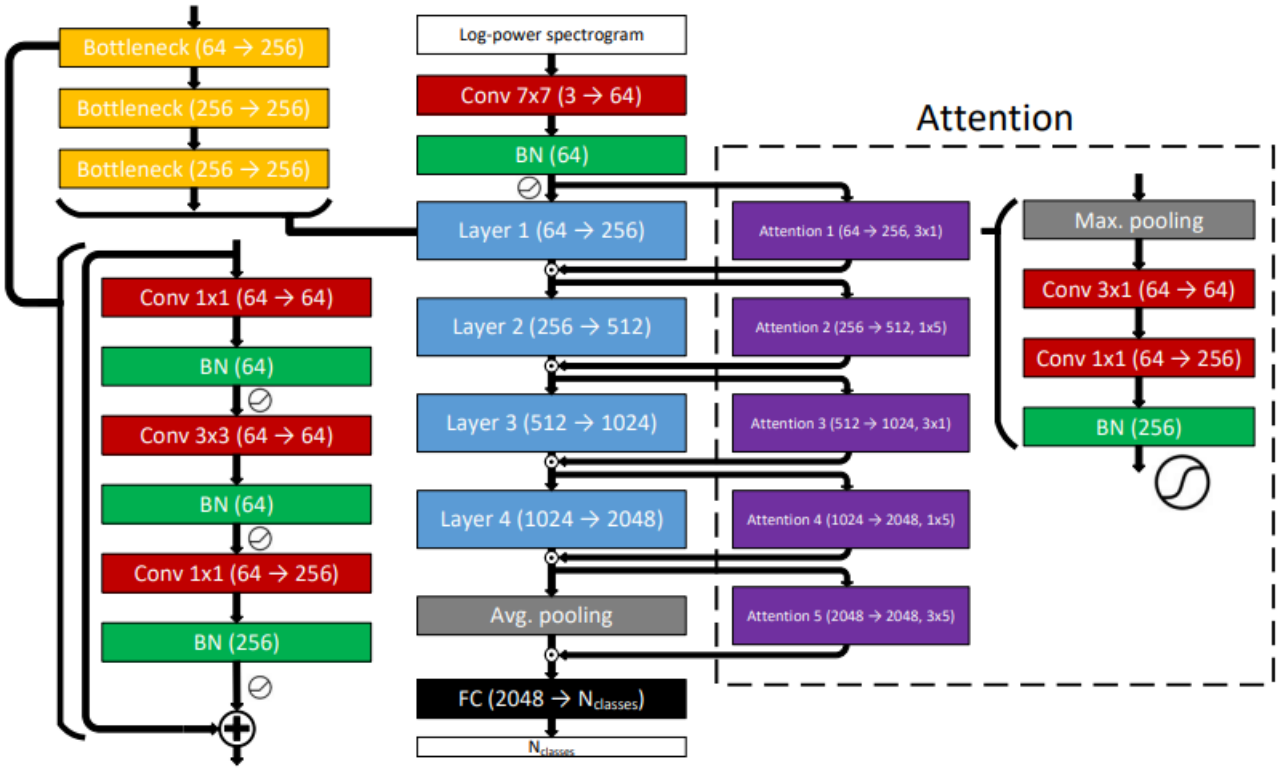
** For each separable, point-wise conv., the abbreviations stand for: *'s'*: shared in stack dimension, *'d'*: shared in dilation dimension, *'a'*: share in both dimension, *'n'*: no sharing operation

4) ESResNet

Цель – классификация звуков (плач ребёнка, рёв двигателя автомобиля)

Существуют несколько моделей архитектуры, рассмотрим модель, обрабатывающий одноканальный ввод(монозвук).

Архитектура:

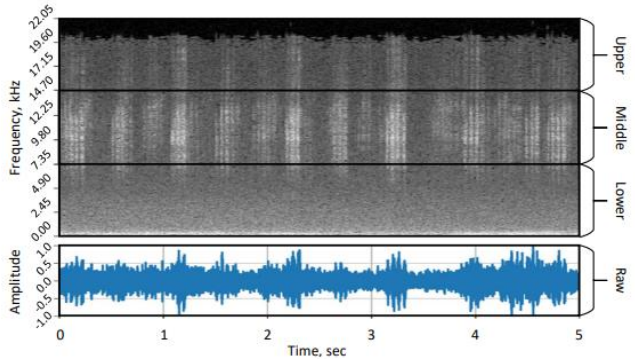


Основная ветвь (2–й столбец) состоит из сверточного слоя (красный), сложенного вместе со слоем пакетной нормализации (зеленый), за которым следуют остаточные слои 1-4 (синий), объединение (серый) и полносвязный слой (черный). Слева представлена типичная структура остаточного слоя. Каждый остаточный слой состоит из стека слоев узких мест (оранжевый), которые включают последовательно применяемые операции Conv-BN и пропускное соединение. ReLU служит функцией активации. Справа (ограниченная пунктирной линией) представлено дополнительное расширение модели ESResNet attention-блоками. Если применяется, блок внимания (фиолетовый) укладывается параллельно остаточному слою 1-4 или среднему слою объединения. Блок внимания включает операцию максимального объединения (серый), за которой следует разделяемая по глубине свертка, сложенная вместе с нормализацией.

Что же касается attention-блока, то для задачи классификации шума основная цель блоков внимания состоит в том, чтобы сфокусировать модель на наиболее важной информации как во временной, так и в

частотной области. Чтобы реализовать механизм attention, в модель ESResNet добавили параллельный стек блоков внимания, где каждый блок из первых обрабатывает информацию, связанную с частотой или временем. Например, первый блок внимания A1 получает тот же входной сигнал, что и первый уровень L1, затем он обрабатывает сигнал x с использованием выделенных по частоте сверточных фильтров и обеспечивает выходной сигнал той же формы, что и у L1. Наконец, вход второго уровня строится путем поэлементного умножения выходов блоков L1 и A1

Обработку входного сигнала можно визуализировать следующим образом (входная волка конвертируется в 3 частотных диапазона, имея в итоге 3 канала):



Данная модель, обучаясь на датасете с 50 классами и 1600 примерами имеет следующие результаты:

Model	Source	Representation	ESC-10	ESC-50	US8K official	US8K unofficial
Human (2015)	[3]	–	95.70	81.30	–	–
Raw waveform and 1D-CNN						
EnvNet (2017)	[5]	raw	88.10	74.10	71.10	–
EnvNet v2 (2017)	[6]	raw	91.30	84.70	78.30	–
Multiresolution 1D-CNN (2018)	[7]	raw	–	75.10	–	–
Gammatone 1D-CNN (2019)	[8]	raw	–	–	–	89.00 [1]
Learnable filterbank and 2D-CNN						
Piczak-CNN + ConvRBM (2017)	[9]	FBE	–	86.50	–	–
Time-frequency representation and 2D-CNN						
Piczak-CNN (2015)	[10]	Mel-spec	90.20	64.50	73.70	–
SB-CNN (2017)	[12]	Mel-spec	–	–	79.00	–
GoogLeNet (2017)	[15]	Mel-spec, MFCC, CRP	86.00	73.00	–	93.00 [2]
Piczak-CNN (2017)	[18]	(TEO-)GT-spec	–	81.95	–	88.02 [3]
Piczak-CNN (2017)	[19]	(PE)FBE	–	84.15	–	–
VGG-like CNN + mix-up (2018)	[21]	Mel-, GT-spec	91.70	83.90	83.70	–
VGG-like CNN + Bi-GRU + att. (2019)	[22]	GT-spec	94.20	86.50	–	–
TSCNN-DS (2019)	[24]	Mel-spec, MFCC, CST	–	–	–	97.20
LMCNet (2019)	[24]	Mel-spec, CST	–	–	–	95.20
LMCNet (no aug.)	*reproduced* [4]	Mel-spec, CST [5]	–	–	*74.04*	*94.00*
TFNet (2019)	[27]	Mel-spec	95.80	87.70	–	88.50
TFNet (no aug.) (2019)	[27]	Mel-spec	93.10	86.20	–	87.20
TFNet (no aug.)	*reproduced* [6]	Mel-spec [7]	–	*79.45*	*78.50*	*96.69*
ESResNet						
from scratch		log-power spec	92.50	81.15	81.31	(96.74)
ImageNet pre-trained		log-power spec	96.75	90.80	84.90	(98.18)
ESResNet-Attention						
from scratch		log-power spec	94.25	83.15	82.76	(96.83)
ImageNet pre-trained		log-power spec	**97.00**	**91.50**	**85.42**	**(98.84)**

Заключение

Нейронные сети произвели революцию во многих областях машинного обучения, обеспечив сверхчеловеческую точность для сложных задач распознавания изображений, выделения звуков и т.д. С ростом популярности голосовых помощников, многие из которых используют методы глубокого обучения, в настоящее время наиболее очевидной задачей в области аудио, вероятно, является автоматическое распознавание речи. Однако, помимо этого очень яркого примера, в области аудио все еще существует множество других проблем, решение которых стало активно появляться только с недавнего времени.

Однако стремление повысить точность часто обходится дорого: современные сети требуют больших вычислительных ресурсов, выходящих за рамки возможностей многих мобильных и встроенных приложений, тут и приходят свёрточные нейронные сети и их подтип – разделяемые по глубине сверточные нейронные сети, который продолжают развиваться и на данный момент являются лучшими решениями в вопросах классификации изображений и звуков и выделения звуков.

Используемая литература

1) https://towardsdatascience.com/types-of-convolutions-in-deep-learning-717013397f4d
2) SliceNet(http://arxiv-export-lb.library.cornell.edu/pdf/1706.03059)
3) PydMobileNet(https://www.researchgate.net/publication/334854007_PydMobileNet_Pyramid_Depthwise_Separable_Convolution_Networks_for_Image_Classification)
4) Xception (https://arxiv.org/abs/1610.02357)
5) MobileNet(https://arxiv.org/abs/1704.04861)
6) SwishNet(http://arxiv-export-lb.library.cornell.edu/pdf/1812.00149)
7) TseNet(http://arxiv-export-lb.library.cornell.edu/pdf/2004.14762)
8) MiTAS(http://arxiv-export-lb.library.cornell.edu/pdf/1912.03884)
9) ESResNet(http://arxiv-export-lb.library.cornell.edu/pdf/2004.07301)