Отчет по заданию SHARE CV «research»[1]

Михальчук Матвей [2]

(МГУ им. М.В, Ломоносова, механико-математический факульет,
каф. высшей геометрии и топологии, гр. 403)

Задание:

Обучить модели на датасете CIFAR100 с ограниченным числом параметров. Необходимо провести исследование и построить кривую в осях количество параметров - качество.
Минимальный набор точек для исследования по числу параметров: 1000, 5000, 10000, 50000, 100000, 500000, 1000000.

---

1 Введение

Для выполнения задания необходимо было выбрать несколько архитектур нейросетей с заданным количеством параметров. При этом выбирать нужно было хорошие, эффективные модели для заданных ограничений (иначе полученная кривая для случайных моделей никакого смысла нести не будет).

Меня всегда интересовал вопрос «как подобрать (оптимальную/хорошую) архитектуру модели», не очень понятно, как эту задачу решать. Конечно, есть ряд подходов, решений, которые успешно сработали в других работах, есть NAS (про который я пока очень мало знаю, но очень интересно). Но на данный момент я плохо понимаю, как подобрать архитектуру модели, близкую к оптимальной, в более-менее общем случае.

И в этом контексте, была идея (описана ниже) поиска оптимальной архитектуры модели из заданного семейства, которую давно хотелось проверить.

[1] Код программ, данный текст и прочие материалы расположены в репозитории по адресу `https://github.com/MatthewMih/small_projects/tree/main/SHARE_research`

[2] mikhalchuk.matvey@gmail.com, tg: @matveymih

Возможно, я несколько отклонился от основной тематики задания, но мне было очень интересно, будет ли работать данный метод, и в случае если он сработает, план был – найти с помощью него хорошие архитектуры моделей (из некоторого семейства архитектур) для различного количества параметров, построить требуемую кривую.

При этом было еще интересно проверить, будут ли какие-либо зависимости между гиперпараметрами моделей (количство каналов, слоев и т.п.) при масштабировании моделей из семейства, похожие на зависимости, описанные в статье про EfficientNet[1]

2 Описание метода

[Дисклеймер: метод примитивный, давно есть более современные NAS-методы, но было очень интересно проверить, сработает ли такая простая идея. Spoiler: не сработала]

Рассмотрим следующую подзадачу: найти архитектуру модели $\mathcal{F}$, показывающую после обучения наибольший accuracy при заданном количестве параметров N.

Конечно, модели будем рассматривать не произвольные, а из некоторого параметрического семейства моделей F (каждая архитектура описывается некоторыми гиперпараметрами $q_1, \ldots, q_n$ (например количеством сверточных слоев, каналов и т.п.))

Т.е. получаем следующую задачу (дискретной) оптимизации

$$\text{Accuracy}(\mathcal{F}(q_1, \ldots, q_n)) \to \max,\ N(q_1, \ldots, q_n) = \text{const},\ \mathcal{F} \in F.$$

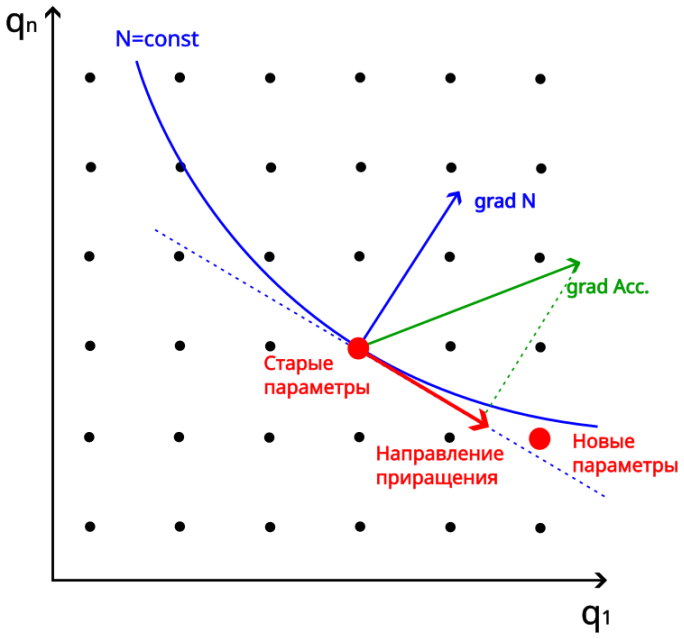
Понятно, что полный перебор всех моделей из семейства вычислительно неэффективен, на это потребуется слишком много времени для больших семейств моделей F.

Однако, мы можем довольно быстро узнать, как изменяется accuracy в локальной области, окрестности точки пространства гиперпараметров. Поэтому попробуем реализовать жадный алгоритм поиска.

Если бы гиперпараметры $q_1, \ldots, q_n$ были непрерывными, то для происка максимума (локального) функции accuracy$(q_1, \ldots, q_n)$ можно было бы использовать итеративный алгоритм – на каждом шаге изменять вектор $\vec{q}$ в направление наибыстрого роста accuracy по гиперповерхности $N(q_1, \ldots, q_n) = const$.

Для этого можно на каждом шаге изменять гиперпараметры на вектор $\delta\vec{q}$, направленный вдоль проекции градиента accuracy на касательную плоскость к поверхности $N = const$:

$$\delta\vec{q} = \eta \cdot \frac{\text{ort}_{\vec{\nabla}N}(\vec{\nabla}Acc.)}{||\text{ort}_{\vec{\nabla}N}(\vec{\nabla}Acc.)||}, \quad \eta = const$$



$$\text{ort}_{\vec{\nabla}N}(\vec{\nabla}acc.) := \vec{\nabla}Acc. - \frac{< \vec{\nabla}Acc., \vec{\nabla}N >}{< \vec{\nabla}N, \vec{\nabla}N >}$$

(Описанный алгоритм – это градиентный спуск по поверхности)

Но в нашей задаче все параметры $q_1, \ldots, q_n$ дискретные, поэтому заменим градиенты на их разностный аналог.

$$\frac{\partial Acc.}{\partial q_i} \rightsquigarrow Acc.(q_1, \ldots, q_i + 1, \ldots, q_n) - Acc.(q_1, \ldots, q_i, \ldots, q_n).$$

Вообще говоря, не понятно, на сколько корректна такая замена, потому что $A(q_1, \ldots, q_n)$ – дискретная функция. Но в работе про EficcientNet [1] авторы смогли найти гладкие монотонные зависимости между оптимальными гиперпараметрами модели от N, поэтому, возможно, функция $Acc.(\vec{q})$ довольно регулярно (монотонно, не быстро меняется) зависит от гиперпараметров, и ее получится оптимизировать предложенным алгоритмом.

Реализация алгоритма

Для реализации алгоритма осталось определить, как измерять $Accuracy(\vec{q})$ и задать семейство архитекрур $F = \{\mathcal{F}(\vec{q})\}$.

Для данного алгоритма я использовал довольно простые сверточные архитектуры, состоящие из блоков сверток и max-pooling-слоев.

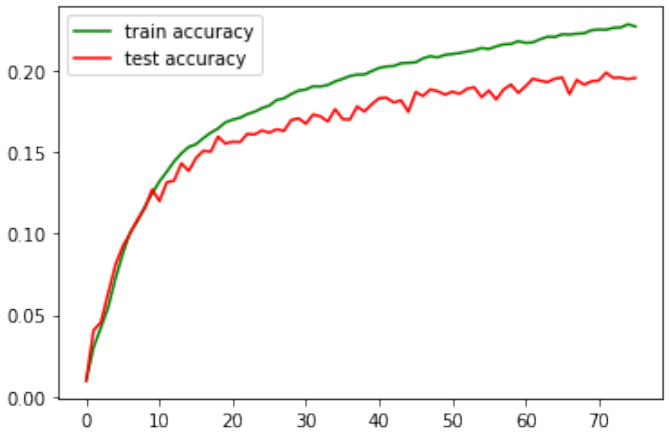
Каждый сверточный блок состоит из нескольких (гиперпараметр k) подряд идущих сверток (conv2d в режиме «SAME», kernel=3), слоев BatchNorm и функций активации Relu.

Сверточный блок (k1, c_in=3, c_out=c1)
MaxPool2d(kernel_size=2)
Сверточный блок (k2, c_in=c1, c_out=c2)
MaxPool2d(kernel_size=2)
Сверточный блок (k3, c_in=c2, c_out=c3)
MaxPool2d(kernel_size=2)
Сверточный блок (k4, c_in=c3, c_out=c4)
AvgPool, Flatten, Linear, Softmax

Таблица 1: нейросеть с гиперпараметрами (k1,k2,k3,k4,c1,c2,c3,c4)

В качестве значения $Accuracy(\vec{q})$ использовалось значение accuracy, полученное на тестовых данных после 75-й эпохи.

Хорошей новостью было то, что случайная ошибка измерения accuracy для данных параметров $\vec{q}$ была меньше, чем изменение accuracy при выборе других гиперпараметров $\vec{q}$.



(Типичная зависимость accuracy от количества эпох.)

Результаты работы алгоритма

Для данного семейства моделей (с последовательными свертками) самой эффективной оказалась модель, в которой в каждом из блоков была только одна свертка (k1=k2=k3=k4=1), поэтому семество было несколько неудачным. Алгоритм работал плохо (результат работы похож на случайные блуждания).

Поэтому данные для решения задачи собирались с помощью архитектур, подобранных вручную (об этом ниже).

Идеи по улучшению алгоритма

Можно попробовать иначе измерять Accuracy: аппроксимировать зависимость acc. от количества эпох t кривой (например $acc.(t) = \alpha - e^{-\beta(t-t_0)}$) и принимать в качестве $Acc.(\vec{q})$ значение, к которому кривая асимптотически стремится.

Выбрать другое семество моделей (использованное было выбрано из-за простоты и недостатка времени, но работают такие модели плохо (при увеличении глубины), как минимум нужно добавить residual-connections...).

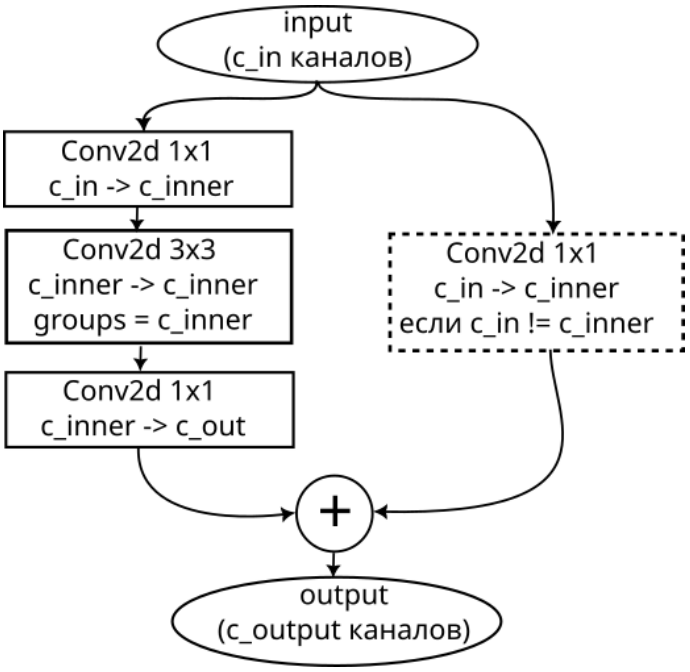
Как были получены требуемые в задании результаты

Для построения кривой использовались 9 моделей, 5 из которых (модели 4–8) построены по образу EfficientNet-B0 (сама EficientNet не использовалась, т.к. там ококло 11 миллионов параметров).

В каждой из этих моделей использовались MBConv-блоки, предложенные в [2] и использованные в EficcientNet.

В отличие от EficcientNet, применялись упрощенные MBConv-блоки, не содержащие S&E слоев, и использующие функцию ReLU в качестве активации.

Модели 4–8 состоят из 3 Maxpool2d-слоев (каждый уменьшает размер изображения в 2 раза), между которыми находятся 1-2 MBConv-блока, затем применяются Average-pooling, полносвязный слой и Softmax. (см. Приложение)



Блок MBConv(c_in,c_out,c_inner). После каждой свертки левой ветви применяется BatchNorm и функция активации ReLU.

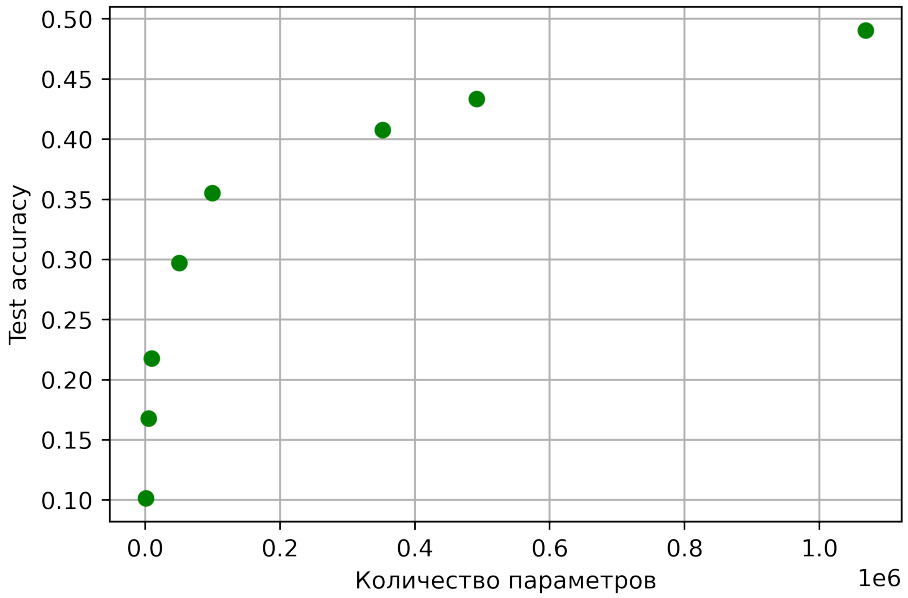
Легкие модели 1-3 не используют MBConv-блоков и residual-connection'ов, это обычные сверточные модели, найденные вручную, в некоторых используются depthwise separable свертки.

Все модели обучались 200 эпох с помощью оптимизатора Adam со стандартными параметрами на видеокарте GeForce GTX 1050 Mobile без аугментации.

Результат обучения моделей

Модель	Количество параметров	Test accuracy	Время обучения
№1	984	0.1014	43 мин.
№2	4 956	0.1678	46 мин.
№3	9 636	0.2177	27 мин.
№4	50 548	0.297	91 мин.
№5	99 844	0.3553	151 мин.
№6	352 388	0.4078	171 мин.
№7	491 524	0.4334	192 мин.
№8	1 068 740	0.4906	242 мин.

Таблица 2: Результат обучения моделей



Что можно улучшить

Есть несколько идей, с помощью которых можно значительно улучшить качество моделей:
1. Подобрать более удачные архитектуры моделей (особенно для маленьких моделей)
2. Использовать прунинг и transfer learning.
3. Аугментация

References

[1] Mingxing Tan, Quoc Le. Efficientnet: Rethinking model scaling for convolutional neural networks. arXiv:1905.11946 , 2019.

[2] Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., and Chen, L.-C. Mobilenetv2: Inverted residuals and linear bottlenecks. CVPR, 2018.

Приложение. Архитектуры моделей

Модель 4: 50548 параметров

nn.Conv2d(3, 16, kernel_size=1),
nn.BatchNorm2d(16),
nn.ReLU(),

MBConv(in_c=16, out_c=16,
inner_c=64),

nn.MaxPool2d(kernel_size=2),
MBConv(in_c=16, out_c=32,
inner_c=64),

nn.MaxPool2d(kernel_size=2),
MBConv(in_c=32, out_c=32,
inner_c=128),

nn.MaxPool2d(kernel_size=2),
MBConv(in_c=32, out_c=100,
inner_c=128),

nn.AvgPool2d(kernel_size=4),
nn.Flatten(),
nn.Linear(100, 100),
nn.Softmax(dim=-1)

Модель 5: 99844 параметров

nn.Conv2d(3, 32, kernel_size=1),
nn.BatchNorm2d(32),
nn.ReLU(),

MBConv(in_c=32, out_c=32,
inner_c=128),

nn.MaxPool2d(kernel_size=2),
MBConv(in_c=32, out_c=32,
inner_c=128),

nn.MaxPool2d(kernel_size=2),
MBConv(in_c=32, out_c=64,
inner_c=128),

nn.MaxPool2d(kernel_size=2),
MBConv(in_c=64, out_c=100,
inner_c=256),

nn.AvgPool2d(kernel_size=4),
nn.Flatten(),
nn.Linear(100, 100),
nn.Softmax(dim=-1)

Модель 6: 352388 параметров

nn.Conv2d(3, 32, kernel_size=1),
nn.BatchNorm2d(32),
nn.ReLU(),

MBConv(in_c=32, out_c=32,
inner_c=128),

nn.MaxPool2d(kernel_size=2),
MBConv(in_c=32, out_c=64,
inner_c=128),

nn.MaxPool2d(kernel_size=2),
MBConv(in_c=64, out_c=128,
inne_c=256),

nn.MaxPool2d(kernel_size=2),
MBConv(in_c=128, out_c=256,
inner_c=512),

nn.AvgPool2d(kernel_size=4),
nn.Flatten(),
nn.Linear(256, 100),
nn.Softmax(dim=-1)

Модель 7: 491524 параметров

nn.Conv2d(3, 32, kernel_size=1),
nn.BatchNorm2d(32),
nn.ReLU(),

MBConv(in_c=32, out_c=32, inner_c=128),

nn.MaxPool2d(kernel_size=2),
MBConv(in_c=32, out_c=64, inner_c=128),

nn.MaxPool2d(kernel_size=2),
MBConv(in_c=64, out_c=128, inner_c=256),

MBConv(in_c=128, out_c=128, inner_c=512),

nn.MaxPool2d(kernel_size=2),
MBConv(in_c=128, out_c=256, inner_c=512),

nn.AvgPool2d(kernel_size=4),
nn.Flatten(),
nn.Linear(256, 100),
nn.Softmax(dim=-1)

Модель 8: 1068740 параметров

nn.Conv2d(3, 32, kernel_size=1),
nn.BatchNorm2d(32),
nn.ReLU(),

MBConv(in_c=32, out_c=32, inner_c=128),

nn.MaxPool2d(kernel_size=2),
MBConv(in_c=32, out_c=64, inner_c=128),
MBConv(in_c=64, out_c=64, inner_c=256),

nn.MaxPool2d(kernel_size=2),
MBConv(in_c=64, out_c=128, inner_c=256),
MBConv(in_c=128, out_c=128, inner_c=512),

nn.MaxPool2d(kernel_size=2),
MBConv(in_c=128, out_c=256, inner_c=512),
MBConv(in_c=256, out_c=256, inner_c=1024),

nn.AvgPool2d(kernel_size=4),
nn.Flatten(),
nn.Linear(256, 100),
nn.Softmax(dim=-1)