Отчет о решении задания второго этапа отбора

Мелехин Денис Антонович

12.04.2021

Описание гипотезы

Для анализа я выбрал вторую статью - “Visual Transformers: Token-based Image Representation and Processing for Computer Vision” (https://arxiv.org/abs/2006.03677)
В данной статье проверяется гипотеза об успешности использования трансформеров в computer vision. Авторы статьи утверждают:
1.Сверточные сети рассматривают все пиксели одинаково, независимо от их важности. Хотя стоило бы приоритизировать объекты первого плана.
2.Сверточные сети стоит применять для поиска низкоуровневых зависимостей, так как они есть на каждом изображении. Но высокоуровневые зависимости, такие как “ухо”, “собака”, “кошка” существуют далеко не на каждом изображении, поэтому вычислительно неэффективно искать их на каждом изображении.
3.Сверточные сети плохо видят зависимости на больших расстояниях. Из-за этого приходится усложнять модель, чтобы компенсировать это недостаток, что так же вычислительно неэффективно

Описание предложенной авторами модели

Для компенсации указанных недостатков сверточных сетей авторы предлагают описывать каждое изображение токенами (так как считают, что любое изображение можно описать несколькими словами). Для получения токенов авторы применяют spatial attention к выходу одному из последних слоев сети, ищущей низкоуровневые зависимости. Далее полученные токены направляются в трансформер, который с помощью self-attention определяет воздействие каждого токена на каждый (т.е. находит высокоуровневые зависимости). Преобразованные последним трансформером токены поступают в обычный модуль классификации.

Проверка гипотезы

Для проверки данной гипотезы я решил воспроизвести эксперименты, предложенные в статье на разных датасетах. К сожалению, только спустя множество тестов я смог понять, что заимствованные мной реализации (прикрепленные к arxiv) содержат баги и отклонений от статьи. Перед каждым экспериментом я буду указывать данные отклонения и особенности теста.

Cifar10

Cifar10 - большой набор изображений, который обычно используется для тестирования алгоритмов машинного обучения. Он содержит 60 000 цветных картинок размером 32x32 пикселя, размеченных в один из десяти классов: самолеты, автомобили, коты, олени, собаки, лягушки, лошади, корабли и грузовики

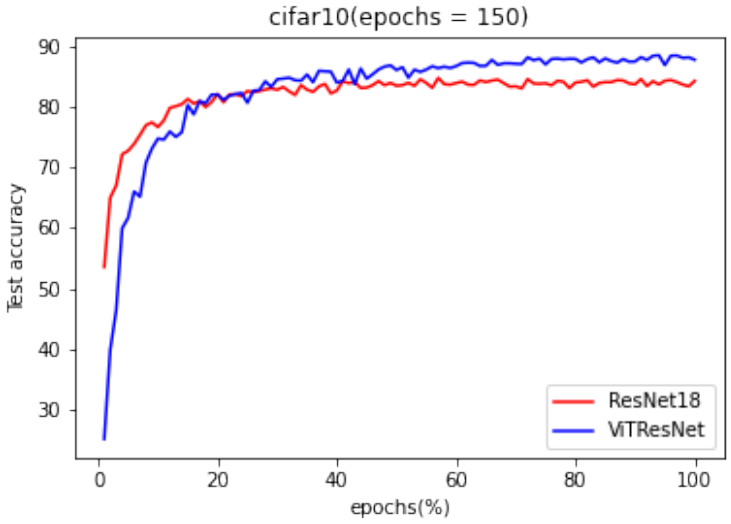
В статье не было тестирований на данном датасете, но мне было интересно узнать, как модель поведет себя на других данных.

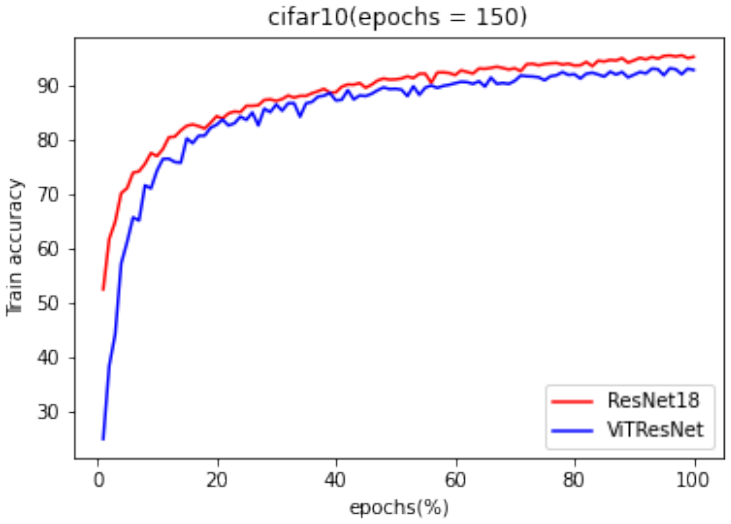
В https://arxiv.org/abs/2006.03677 я нашел реализации методов статьи и решил использовать один из них: https://github.com/tahmid0007/VisualTransformers

Отклонения:
1.Использовался только один Visual Transformer модуль
2.Использовались эмбеддинги между токкенизатором и транформером
3.Так как был только один Visual Transformer модуль, projector естественно не использовался.
4.Оптимизатор Adam, а не SGD как использовалось в статье

Есть еще несколько отклонения от статьи, но идейно модель соответствует статье.

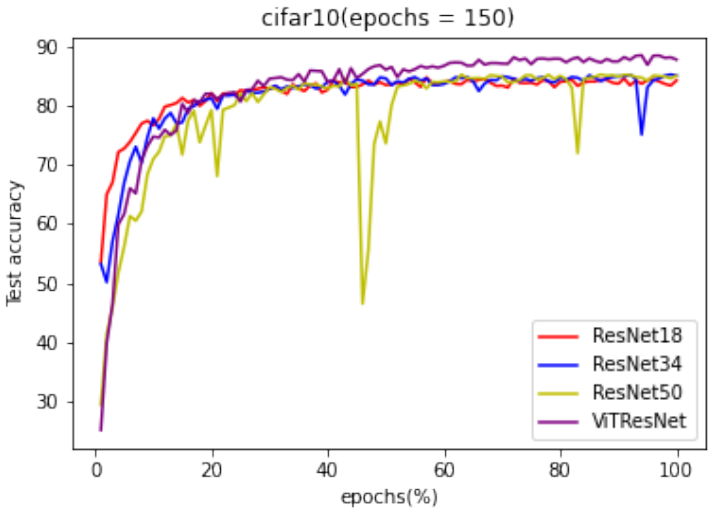
Немного изменив реализацию, я запустил тестирование на 150 эпохах и получил следующие результаты для train и test (оптимизаторы для обоих моделей был идентичным).

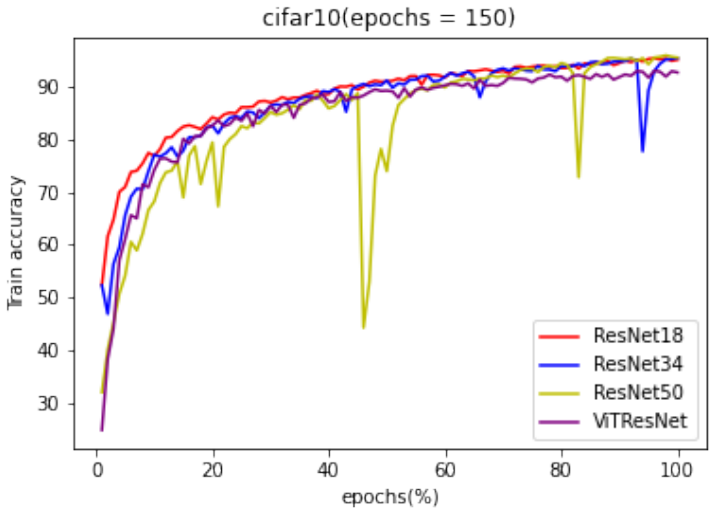




По графикам видим, что классический ResNet18 имеет чуть большую точность по сравнению с ViTResNet на train, но описанная в статье модель довольно существенно обгоняет классический ResNet18 на test.

Возможно, не совсем целесообразно тестировать ResNet34 и ResNet50 на таком, относительно, маленьком датасете, но мне было интересно их поведение, поэтому вот сравнение всех моделей:





На данных графиках видны отличные результаты модели статьи, ведь она обгоняет все классические ResNet на тестовом датасете(но, повторюсь, более глубокие ResNet ведут себя ненамного лучше своих младших братьев, так как размер датасета относительно маленький)

Сравнение производительности (из ноутбука get_flops.ipynb)

ViTResNet: 9601.28 FLOPs(M) 1.06217 Params(M)

ResNet18: 7525.1712 FLOPs(M) 11.689512 Params(M)

ResNet34: 15085.1584 FLOPs(M) 21.797672 Params(M)

ResNet50: 17183.3344 FLOPs(M) 25.557032 Params(M)

Вообще, ViTResNet должен обгонять ResNet18 в производительности, но в данном случае, у него намного больше параметров, но производительность меньше.

Tiny Image Net

База данных ImageNet — проект по созданию и сопровождению массивной базы данных аннотированных изображений. Аннотация изображений происходит путем краудсорсинга сообществом. Из-за этого достигается большое количество размеченных данных.

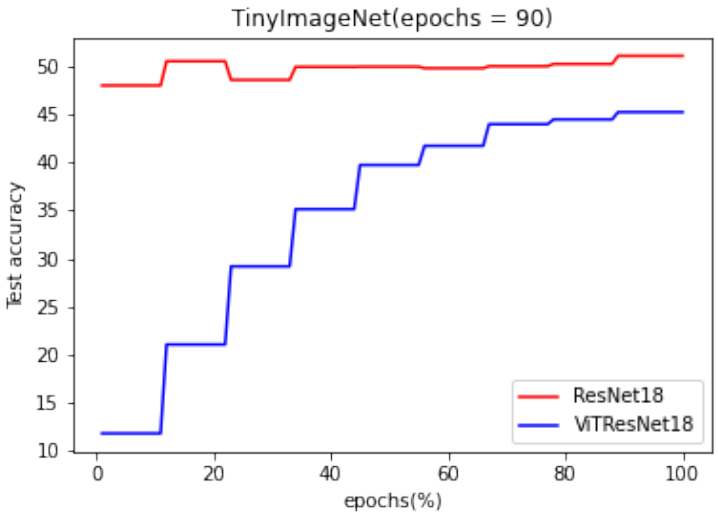
http://cs231n.stanford.edu/tiny-imagenet-200.zip

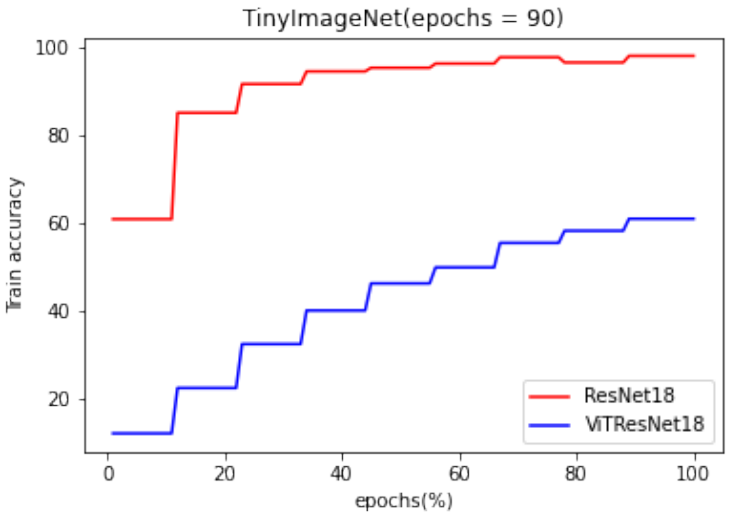
Так как тестирование на обычном ImageNet у меня заняло бы месяцы (а еще он не в открытом доступе), я решил протестировать модели на его подвыборке с 100000 изображениями для обучения, 10000 изображениями на валидации и 200ми классами

изображений. Каждое изображение было с разрешением 64x64. Для предобработки датасета и загрузки его в Data Loader я воспользовался : https://github.com/leemengtaiwan/tiny-imagenet/blob/master/TinyImageNet.py

После нескольких тестов я понял, что необходимо увеличить разрешение изображений до 224x224 (которое требует классическая ResNet)

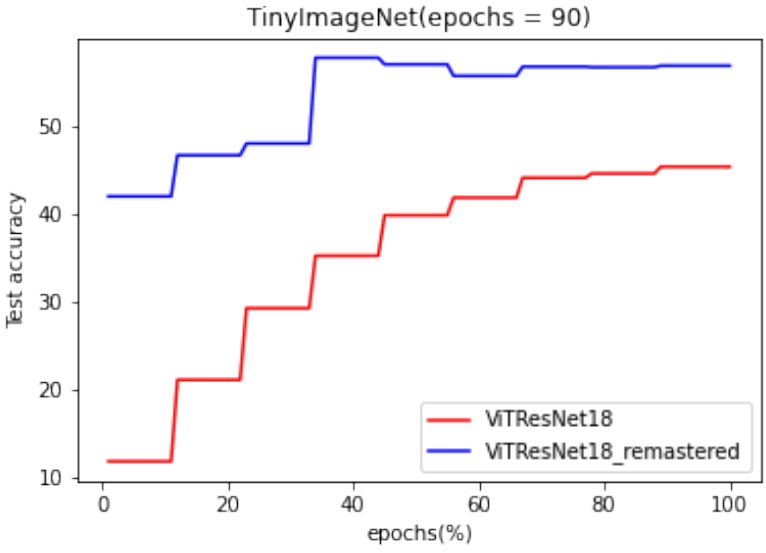
Первое тестирование с первой версией ViTResNet (на гите она будет называться not remastered) с оптимизатором Adam дало следующие результаты:

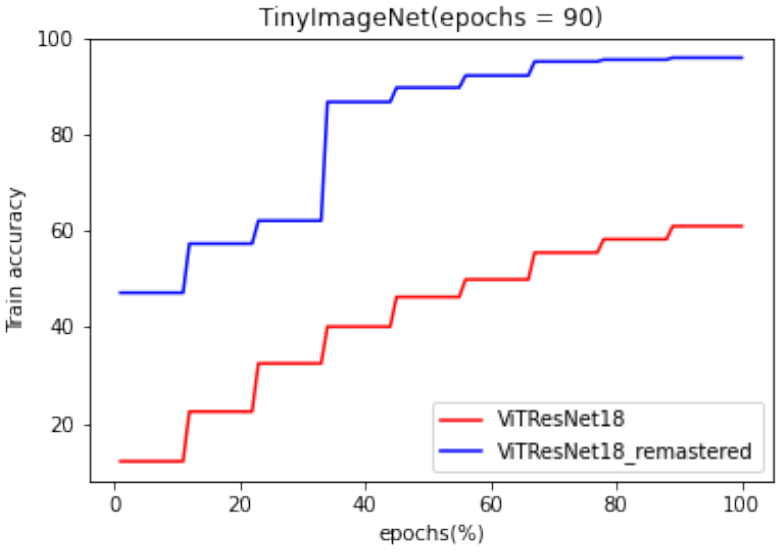




Классическая ResNet в пух и прах разносила ViTResNet. Не поверив в это, я решил почти полностью переписать модель с намного менее значимыми отклонениями от статьи. Я базировался на решениях:
https://github.com/tahmid0007/VisualTransformers
https://github.com/NazirNayal8/visual-transformer

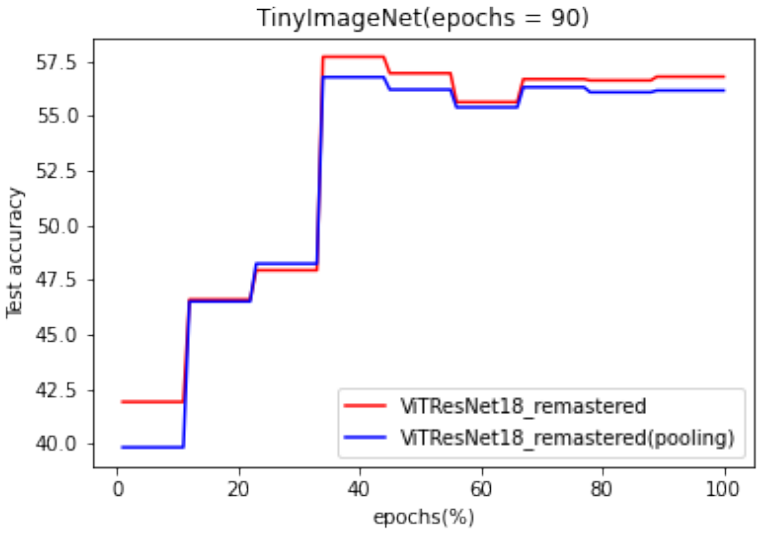
Но от них в переписанной модели осталось мало чего.
Единственное оставшееся отклонение от статьи - в трансформере, который остался почти прежним, но все же очень похожим на описанный в статье трансформер. Remastered ViTResNet18 обучалась уже на оптимизаторе SGD с указанными в статье параметрами

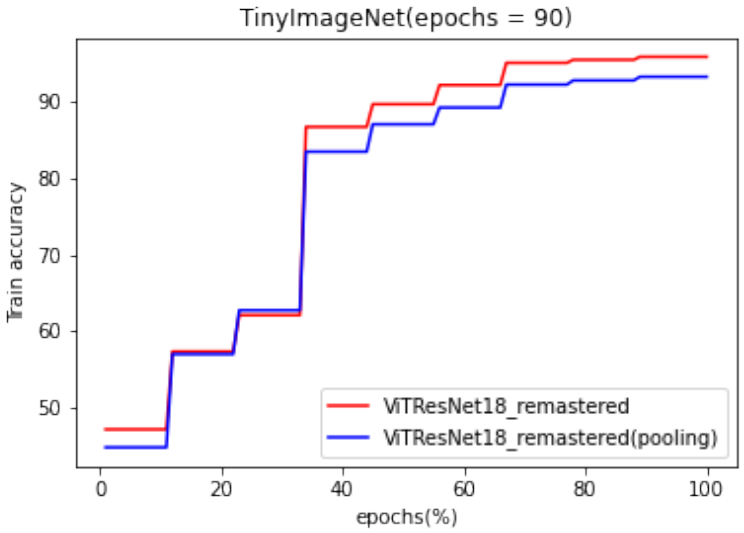




Переписанная модель дает лучшие результаты, по сравнению с заимствованной моделью.

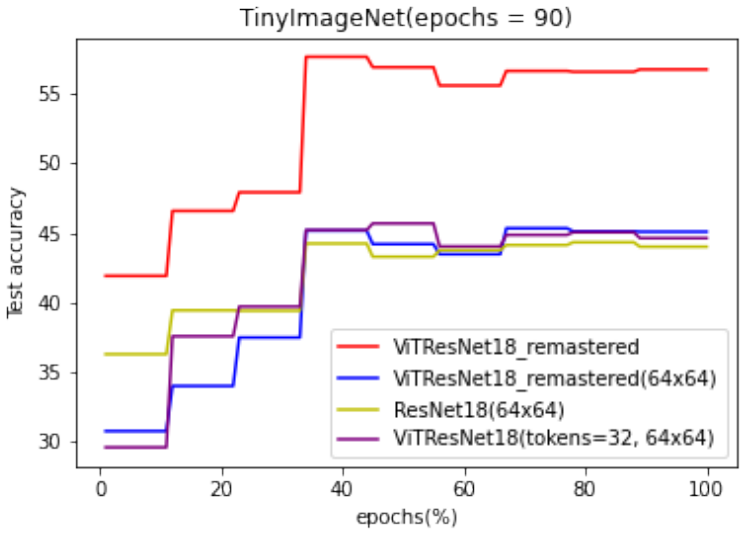
Представленные выше графики для модели с filter based tokenizer. Я решил сравнить их с pooling based tokenizer:

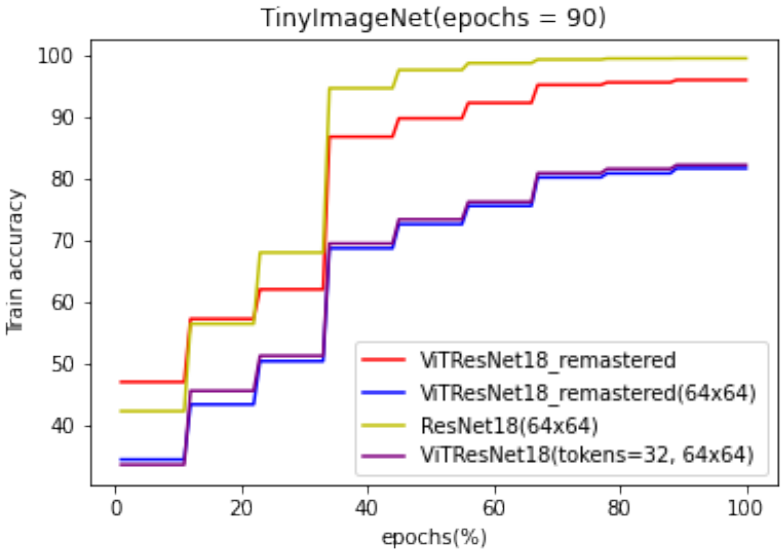




По графикам видно, что filter based tokenizer показывает чуть лучшие результаты, чем Pooling based tokenizer.

Также, для объяснения решения увеличить разрешение изображение до 224x224 представляю график сравнения результатов одной и той же модели, но обученной на изображениях разного разрешения:





Видно, что у модели обученной на изображениях 64x64 точность предсказания на тесте значительно хуже. Также видно, что ResNet18 очень сильно переобучилась. Также этот эксперимент подтверждает гипотезу статье о том, что увеличение числа токенов не

улучшает модель. По графиками видно, что модель с 16ю токенами дает почти такую же точность, как и 32мя.

ResNet18(64x64): 1819.066368 FLOPs(M) 111.689512 Params(M). (Из колаба)

ViTResNet18_remastered: 5700.4544 FLOPs(M) 2.792456 Params(M)

Скорее всего, модель из статьи можно было реализовать более эффективно и из-за этого сейчас она проигрывает в производительности.

Также, я написал рекуррентный токенизатор, но к сожалению, успел обучить его только на 30 эпохах.

Описание гита

Все тесты

Особенности работы

1) Для тестирования в репозиториях, прикрепленных к arxiv использовалась аугментация для train и test. Для теста я убрал изменения самого изображения и оставил только приведение к тензору и нормализацию
2) В статье просилось использовать батч размера 256, но я использовал меньше, так как такой большой батч не влезал в память моей видеокарты
3) Для того, чтобы можно было прерывать обучение и потом продолжать его, я написал сериализатор модели (он в common.py)
4) В некоторых репозиториях прикрепленных к arxiv обучают линейные слои, а не матрицы. Я обучал матрицы, но это идентично.
5) В not remastered не использовался sheduler, какой просился в статье
6) Размер токенов в 1024 (как просилось в статье) давало намного хуже результаты, чем 256. Это из-за того, что Tiny Image Net намного меньше Image Net
7)

Выводы

По проведенными мной экспериментам можно сделать вывод, что визуальные трансформеры РАБОТАЮТ. Они всегда выигрывают в точности предсказания. В моих экспериментах визуальные трансформер проигрывал в производительности, но скорее всего это из-за неэффективной реализации. Так что всем советую визуальные трансформеры!