Diffusion Project

21 декабря 2024 г.

1 Постановка задачи и описание методологии

В проекте мы реализовали метод дистилляции диффузионных моделей Distribution Matching Distillation. Он позволяет обучить одношаговый генератор картинок на основе предобученной диффузионной модели и значительно ускорить сэмплирование.

Его основная идея достаточно прямолинейна: минимизировать KL-дивергенцию между распределением генераций учителя и ученика, чей градиент может быть представлен в виде разности двух score-функций. Score-функция распределения картинок учителя у нас уже есть – это диффузионная модель-учитель, вторую же score-функцию мы учим сами. Также к функционалу для ученика добавляется лосс регуляризации: в данном методе это lpips.

Таким образом, у нас есть две модели: генератор G, из которого мы и будем за один шаг сэмплировать картинки, и μ_{fake}, которая выучивает score-функцию распределения генератора. Посмотрим, как выглядит сама процедура обучения.

Algorithm 1: DMD Training procedure

Input: Pretrained real diffusion model μ_{real}, paired dataset $\mathcal{D} = \{z_{\text{ref}}, y_{\text{ref}}\}$
Output: Trained generator G.

```
// Initialize generator and fake score estimators from pretrained model
```
$G \leftarrow \text{copyWeights}(\mu_{\text{real}})$, $\mu_{\text{fake}} \leftarrow \text{copyWeights}(\mu_{\text{real}})$
while *train* **do**
```
    // Generate images
```
    Sample batch $z \sim \mathcal{N}(0, \mathbf{I})^B$ and $(z_{\text{ref}}, y_{\text{ref}}) \sim \mathcal{D}$
    $x \leftarrow G(z),\ x_{\text{ref}} \leftarrow G(z_{\text{ref}})$
    $x = \text{concat}(x, x_{\text{ref}})$ **if** dataset is LAION **else** x

```
    // Update generator
```
    $\mathcal{L}_{\text{KL}} \leftarrow \text{distributionMatchingLoss}(\mu_{\text{real}}, \mu_{\text{fake}}, x)$ `// Eq 7`
    $\mathcal{L}_{\text{reg}} \leftarrow \text{LPIPS}(x_{\text{ref}}, y_{\text{ref}})$ `// Eq 9`
    $\mathcal{L}_G \leftarrow \mathcal{L}_{\text{KL}} + \lambda_{\text{reg}} \mathcal{L}_{\text{reg}}$
    $G \leftarrow \text{update}(G, \mathcal{L}_G)$

```
    // Update fake score estimation model
```
    Sample time step $t \sim \mathcal{U}(0, 1)$
    $x_t \leftarrow \text{forwardDiffusion}(\text{stopgrad}(x), t)$
    $\mathcal{L}_{\text{denoise}} \leftarrow \text{denoisingLoss}(\mu_{\text{fake}}(x_t, t), \text{stopgrad}(x))$ `// Eq 6`
    $\mu_{\text{fake}} \leftarrow \text{update}(\mu_{\text{fake}}, \mathcal{L}_{\text{denoise}})$
end while

Посмотрим внимательнее на лоссы. distributionMatchingLoss – это аппроксимация градиента той

самой KL-дивергенции между распределениями картинок моделей ученика и учителя через их score-функции:

$$\nabla_\theta D_{KL} \simeq \mathop{\mathbb{E}}_{z,t,x,x_t} \left[w_t \alpha_t \left(s_{\text{fake}}(x_t, t) - s_{\text{real}}(x_t, t) \right) \frac{dG}{d\theta} \right]$$

denoisingLoss – это стандартный лосс для обучения диффузионной модели в μ-параметризации, когда ей на вход подаётся зашумлённая картинка и она пытается предсказать изначальный сэмпл:

$$\mathcal{L}^{\phi}_{\text{denoise}} = ||\mu^{\phi}_{\text{fake}}(x_t, t) - x_0||^2_2,$$

Теперь, когда мы закончили с формальной процедурой обучения, можно поговорить о том, почему этот метод вообще работает, т.е. выученный генератор G действительно за 1 шаг генерирует почти такие же по качеству картинки, как диффузионная модель учитель за несколько десятков шагов сэмплирования. Всё дело в score-функциях, на разность которых мы и сдвигаем параметры генератора. Как известно, score-функция – это производная логарифма плотности, т.е. она показывает направление роста плотности. Поэтому, когда мы обновляем параметры на разность score-функций между достаточно реалистичными выходами модели учителя и сгенерированными нами картинками, мы подталкиваем однашаговые сэмплы генератора к большей правдоподобности, т.е. к большей похожести на картинки учителя, и меньшей "фальшивости".

2 Техническое описание

2.1 Подготовка к обучению

Для эксперимента использовали датасет CIFAR-10. В качестве модели-учителя и архитектуры для обучения использовали EDM-VE предобученную на CIFAR-10. В качестве одношагового генератора взяли EDM-VE обусловленный на уровень шума 100. С помощью предобученной модели и 64-шагового heun солверма сгенерировали датасет пар (шум - сгенерированное изображение). В итоге получили датасет размера 2560.

2.2 Обучение на все лоссы

В качестве оптимизатора использовали AdamW со стандартными параметрами из pytorch и lr = $5e-5$. Так же на первых 500 шагах использовали LinearLR с start_factor = 0.01. Обучение происходило 16000 итераций и заняло около 10 часов на 1 T4 GPU.

К сожалению, на датасфере не хватило места и ноутбук и соответственно график 3 обучения решил не сохраниться (хорошо, что удалось спасти веса модели на последнем шаге). Но далее функция потерь вышла на плато и почти не убывала.

2.3 Обучение без LPIPS

В качестве оптимизатора использовали AdamW со стандартными параметрами из pytorch и lr = $1e-5$. Так же на первых 500 шагах использовали LinearLR с start_factor = 0.01. Обучение происходило 5000 итераций и заняло около 3 часов на 1 T4 GPU.

По графику 2 обучения видно, насколько нестабильным стало обучение.

Рис. 1: График обучения со всеми функциями потерь

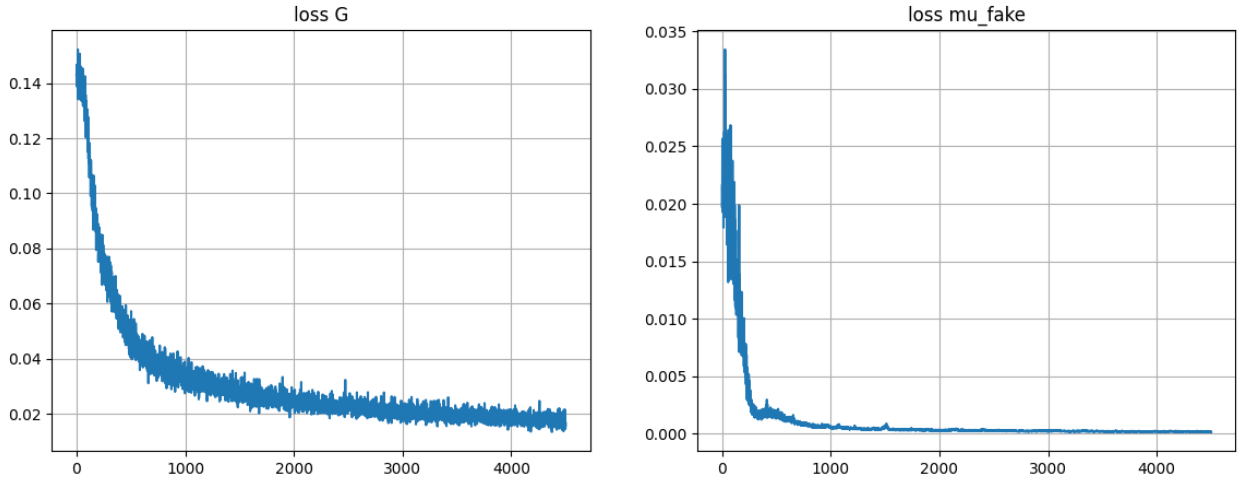
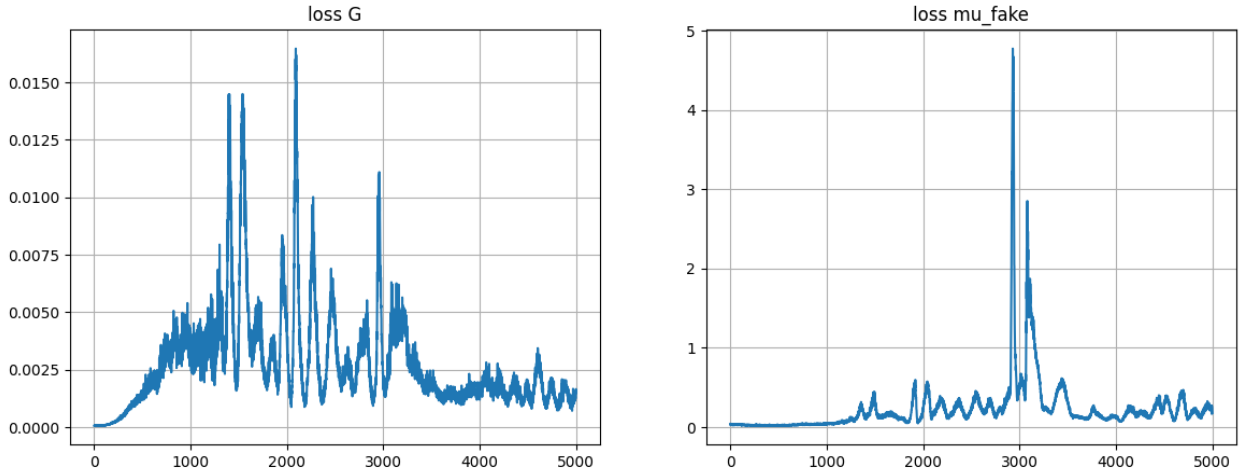


Рис. 2: График обучения без LPIPS

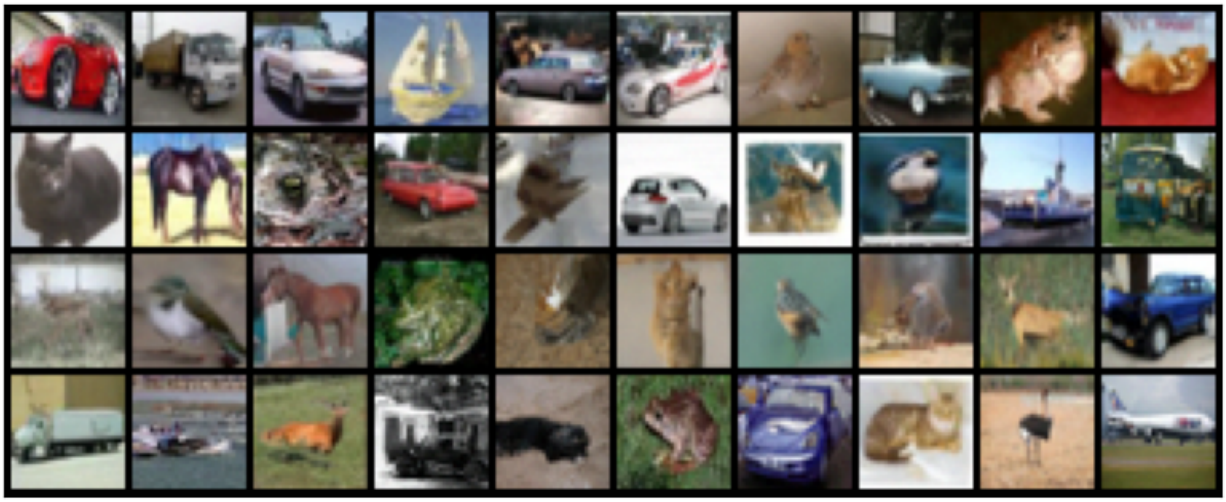


3 Анализ результатов

В процессе анализа мы сравнивали модель учителя с тем семплированием, которое применялось для создания тренировочных пар (heun sampler, num steps = 64), модель ученика, обученную с использованием LPIPS loss, а также модель ученика, обученную без использования этого лосса.

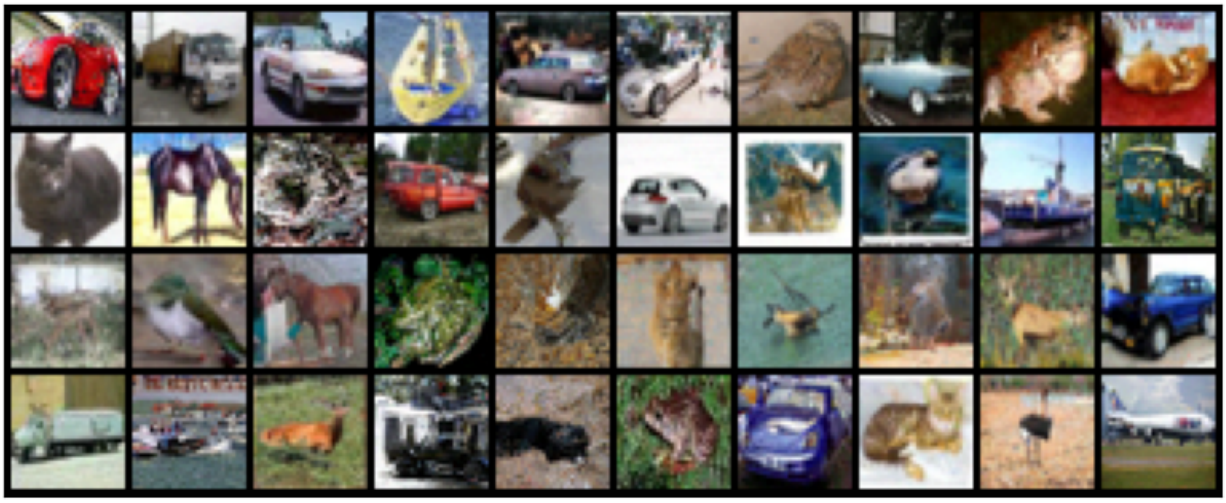
Модель учителя генерирует качественные семлпы при достаточно большом количестве шагов. Пример семплов с количеством шагов 500 (все последующие семплы сгенерированы из одного шума и меток класса):

Teacher model (heun sampler, 500 steps)



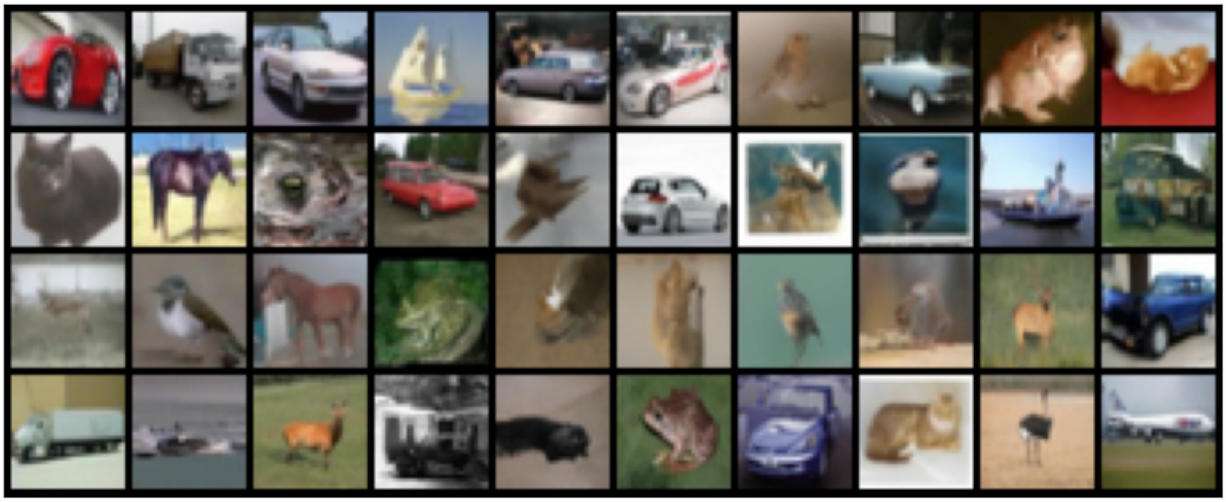
Также мы попробовали использовать euler семплер с тем же количеством шагов. Примеры семплов:

Teacher model (euler sampler, 500 steps)



Визуализируем теперь картинки при 64 шагах, так как такие семплы использовались при составлении тренировочного датасета. Пример семплов с количеством шагов 64 heun sampler:

Teacher model (heun sampler, 64 steps)



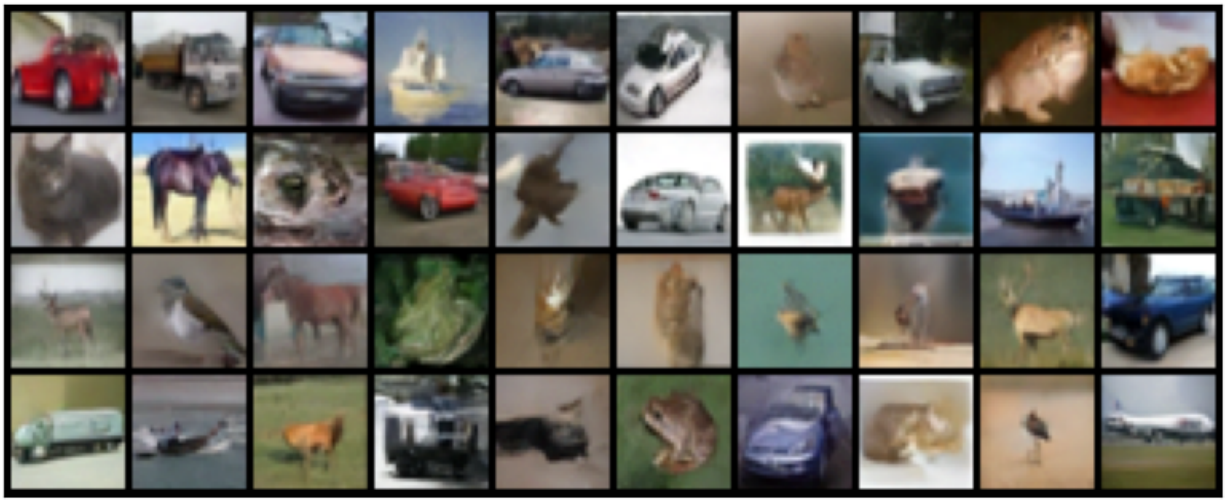
Пример семплов с количеством шагов 64 euler sampler:

Teacher model (euler sampler, 64 steps)

Картинки в обоих методах семплирования при 500 шагах выглядят достаточно качественно, при euler семплировании картинка как будто чуть более “зернистая”, особенно если смотреть на фон, при heun семплировании картинки более гладкие. Однако при 64 шагах разница очень заметна. Картинки при euler семплировании не выделились до конца из шума. При этом при heun семплировании картинки достаточно неплохого качества, хоть и более размытые и с меньшим количеством деталей чем при 500 шагах. Это достигается засчет того что в heun семплировании используется дополнительная информация о векторе направления с предсказанного шага и как мы видим это сильно улучшает аппроксимацию процесса расшумления.

Получившиеся результаты у обученной модели ученика с использованием LPIPS лосса (1 шаговая генерация):

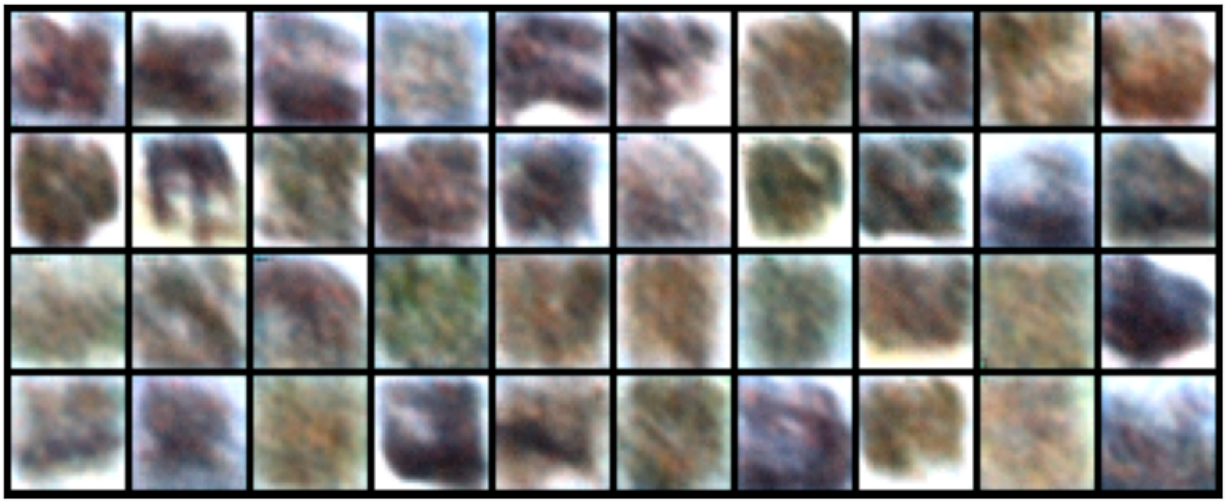
Student model (with LPIPS)



Картинки получились очень близкими к генерации модели учителя, где-то даже практически идентичными, однако в некоторых случаях объект вышел не таким реалистичным например первая машина, и не до конца вырисовались некоторые детали например шея у птицы. Но учитывая что это одношаговая генерация, это очень неплохой результат.

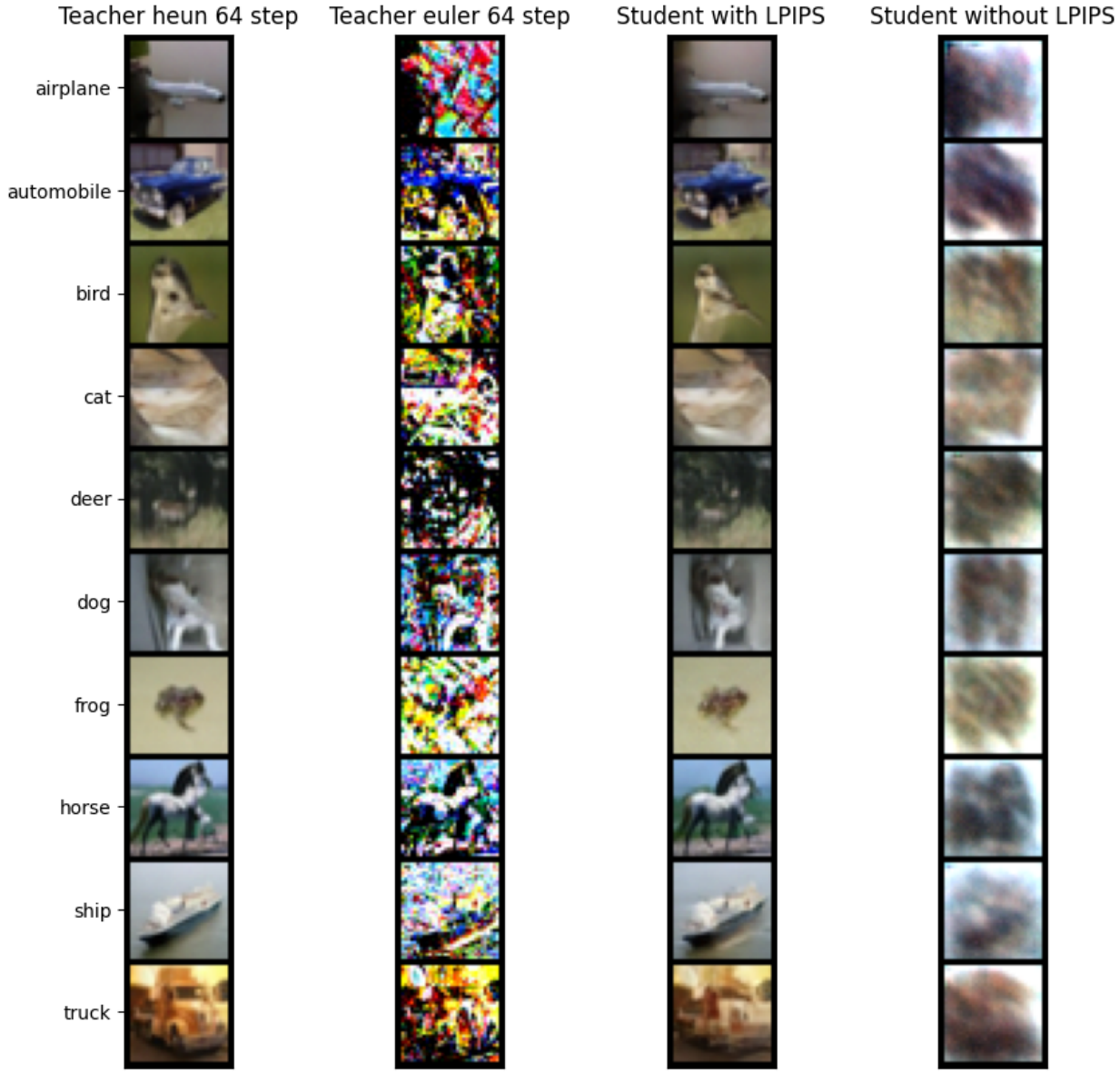
Получившиеся результаты у обученной модели ученика без использования LPIPS лосса (1 шаговая генерация)

Student model (without LPIPS)



Картинки получились размытыми, некачественными, однако если сравнивать с генерациями модели учителя то есть совпадения в цветах и в локализации объекта.

Теперь сравним генерации для каждого из 10 классов:



Опять же тут прослеживаются те же закономерности, стоит отметить что объекты у учителя и ученика (с lpips) получаются одни и те же.

Почему возможно обучение без LPIPS получилось нестабильным а генератор некачественным?

1. Недостаточно учится фейковая диффузионная модель, в статье Improved Distribution Matching Distillation for Fast Image Synthesis утверждается что если 5 итераций учить фейковую диффузионку на 1 итерацию обновления весов ученика, то обучение будет стабильным и LPIPS нам

не нужен. Поэтому возможно в нашем случае недостаточное обучение фейковой диффузионной модели сделало обучение очень шумным а регуляризатор в виде lpips мы убрали.

2. Изначально идея DMD исходила из того что мы хотим чтобы распределения генераций учителя и ученика совпадали, но из-за нестабильности/неопределенности скор функции при около нулевом зашумлении мы перешли к близости зашумленных распределений, оптимумы этих задач действительно совпадают но возможно такой лосс сложнее оптимизировать без регуляризатора, который как раз-таки был направлен именно на выходы модели учителя и ученика, на увеличение их схожести.

Метрика FID показывает насколько отличаются два распределения картинок, для этого берутся наборы семплов из одного распределения картинок (сгенерированных) и другого распределения (реальных данных), картинки прогоняются через предобученный классификатор и далее предполагается что полученные латентные признаки из многомерного нормального распределения, и fid между ними рассчитывается следующим образом:

$$\mathrm{FID}(r, g) = ||\mu_r - \mu_g||^2 + \mathrm{Tr}(\Sigma_r + \Sigma_g - 2(\Sigma_r \Sigma_g)^{1/2})$$

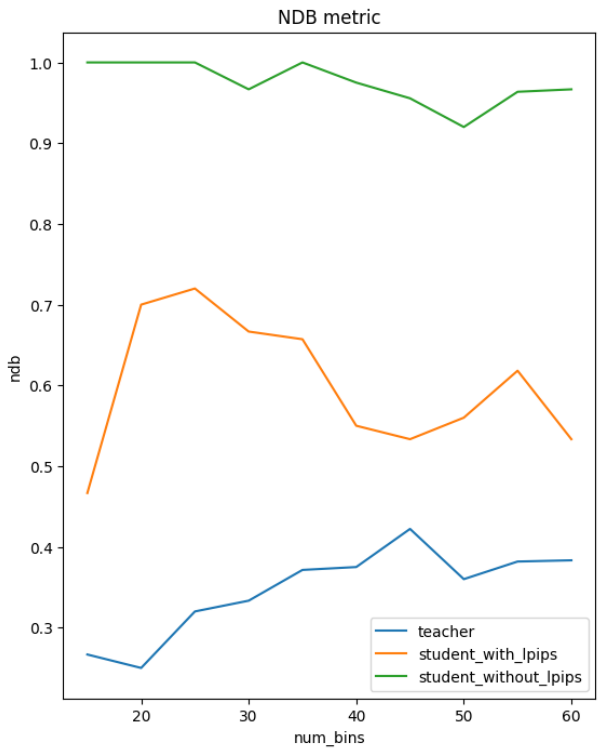
Стоит отметить что FID достаточно чувствителен к размеру выборок. Мы оценивали FID на сгенерированных семплах из 8700 картинок. И при сравнении с распределением реальных данных использовали статистики для CIFAR датасета из edm. Результаты:

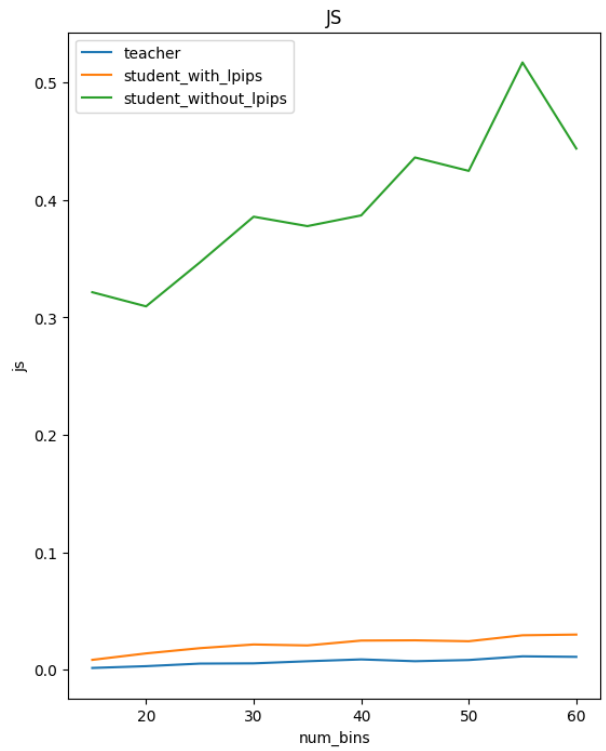
	FID
Teacher model vs real data	8.3
Student with LPIPS vs real data	15.07
Student without LPIPS vs real data	221
Student with LPIPS vs teacher model	10.43
Student without LPIPS vs teacher model	218

Как можно заметить, дистиллированная модель имеет больший FID относительно реальных данных, чем исходная модель, а также распределения учителя и ученика ближе чем распределения ученика и реальных данных что логично особенно в ситуации с lpips, так как регрессионная часть в лоссе способствует тому что при одинаково взятом шуме полученные картинки из модели учителя и ученика в нашем векторном пространстве должны быть близки. Также FID очень большой у модели ученика без lpips лосса что подтверждает численно некачественность семплов.

Также проанализируем NDB & JS метрики используемые для оценки качества сгенерированных данных. Суть метода: латенты реальных данных из датасета мы кластеризуем на какое-то количество бинов, далее мы считаем долю попавших в каждый бин объектов, а также распределяем по этим бинам латенты сгенерированных данных и также считаем долю попавших в каждый бин объектов. Далее метрика NDB вычисляется как доля значимо различных бинов, а JS вычисляет расстояние (дивергенцию Йенсона-Шеннона) между двумя распределениями по бинам. Метрика NDB и JS хорошо отражает проблему mode collapse, когда какие-то моды распределения игнорируются генератором.

Результаты:





У модели ученика без lpips в целом плохие генерации поэтому высокое значение NDB и JS метрик логично. Что касается модели ученика с lpips лоссом, у него от 55-67% бинов отличаются статистически. В то время как у модели учителя от 30-40% бинов что говорит о неком изменении в распределении мод в модели ученика относительно модели учителя. Однако дивергенция Йенсена-Шеннона дополняет картину, и показывает что хоть и распределение по бинам отличается у модели ученика больше чем у модели учителя, они в целом достаточно близки.

Метрика precision&recall, density&coverage вычисляют соответственно точность генератора и полноту покрытия распределением генератора реального распределения:

$$\text{precision} := \frac{1}{M}\sum_{j=1}^{M}\mathbf{1}_{Y_j\in\text{manifold}(X_1,\cdots,X_N)}$$

$$\text{recall} := \frac{1}{N}\sum_{i=1}^{N}\mathbf{1}_{X_i\in\text{manifold}(Y_1,\cdots,Y_M)}$$

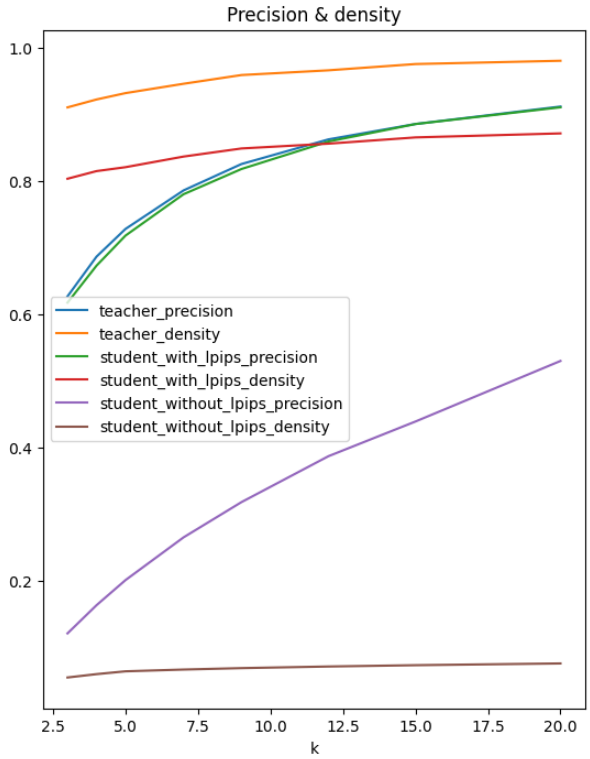
$$\text{density} := \frac{1}{kM}\sum_{j=1}^{M}\sum_{i=1}^{N}\mathbf{1}_{Y_j\in\mathcal{B}(X_i,\text{NND}_k(X_i))}$$

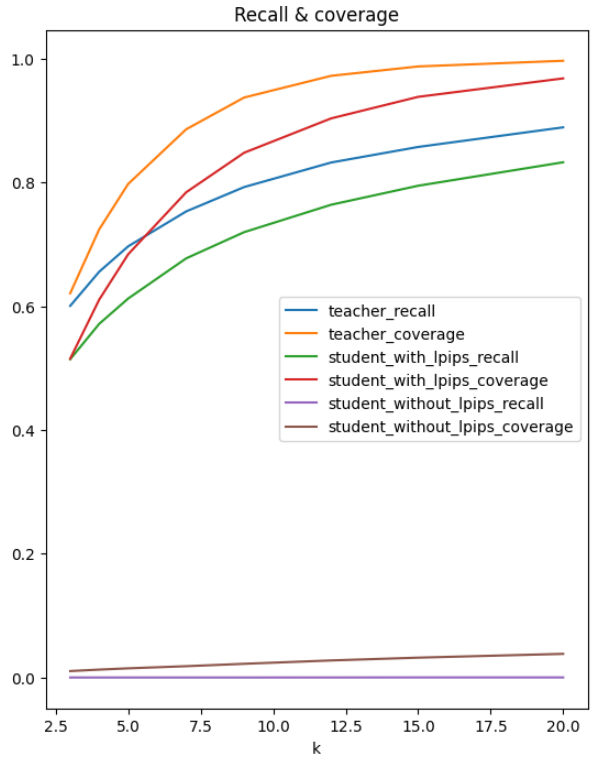
$$\text{coverage} := \frac{1}{N}\sum_{i=1}^{N}\mathbf{1}_{\exists j\text{ s.t. }Y_j\in\mathcal{B}(X_i,\text{NND}_k(X_i))}$$

$$\text{manifold}(X_1, \cdots, X_N) := \bigcup_{i=1}^{N} \mathcal{B}(X_i, \text{NND}_k(X_i))$$

Precision оценивает вероятность появления сгенерированного объекта в многообразии реальных данных, а recall оценивает вероятность наличия реальных выборок в многообразии сгенерированных данных. Density оценивает сколько real data neighbourhood spheres содержат в среднем сгенерированный объект, то есть насколько генерации плотно сосредоточены среди реальных данных. Coverage оценивает долю реальных данных, чьи neighbourhoods содержат хотя бы один сгенерированный семпл.

Результаты:





Таким образом мы получили что точность генераций и у модели учителя и у модели ученика одинаковая. При этом плотность сгенерированных данных по отношению к реальным у учителя больше чем у ученика (у учителя на уровне 0.9, у ученика на уровне 0.8). Recall и coverage тоже немного больше у учителя чем у ученика, что значит что у учителя лучше покрытие реального датасета.

PPL - метрика оценки гладкости латентного пространства, она измеряет, насколько небольшие изменения в латентном пространстве приводят к небольшим изменениям в пространстве эмбеддингов изображений. Косвенно эта метрика может отражать степень разнообразия генераций. Результаты:

	PPL
Teacher	0.00025691945558130904
Student with LPIPS	0.007416443485638472
Student without LPIPS	0.007725048977688659

Видно что у ученика в целом большее значение что значит что при одинаковом изменении шума выходы студента могут быть более разнообразны.

4 Выводы

Таким образом DMD метод для дистилляции диффузионных моделей показал себя очень хорошо, за 1 шаг генерации (а то есть в 64 раза быстрее чем референсная генерация из модели учителя) модель выдает достаточно хорошие визуально семплы, а также по метрикам качества лишь немного уступает учителю. Однако добавление LPIPS оказалось значимым в обучении данной модели, без использования этого лосса модель не сходилась и генерировала очень плохие картинки. Минусы данного подхода к дистилляции: усложненное обучение за счет дополнительной фейковой диффузионной модели, получаем модель для one-to-one генераций, то есть конкретному семплу шума соответствует конкретная картинка.

5 Распределение задач

1. Курдун Мария - анализ качества моделей, их сравнение, подсчет метрик
2. Сенкевич Евгений - обучение моделей, построение датасета, помогал с дебагом функций потерь
3. Тяжова Наталья - реализация функций потерь и шага обучения