

Deep Learning Homework

Pavel Fakanov

15 декабря 2020 г.

1 Введение

Данное практическое задание посвящено реализации и обучении нейросети для перевода изображения в другую модальность. В рамках этого задания решается задача колоризации. Были проведены эксперименты на двух наборах данных, о которых будет рассказано далее.

2 Постановка задачи

По входной черно-белой картинке хочется получить цветную версию данной картинки.



Рис. 1: Колоризация изображения

3 Generative Adversarial Network

3.1 Vanilla GAN

Математические выкладки и детально описание оставим за пределами данного отчета, так как они не являются первичной целью этого домашнего задания. Предоставим лишь общее описание используемого подхода, более детальное описание может быть найдено в оригинальных статьях.

Суть идеи в комбинации двух нейросетей, при которой одновременно работает два алгоритма “генератор” и “дискриминатор”. Задача генератора – генерировать объекты заданной категории. Задача дискриминатора – пытаться распознать сгенерированные объекты.

В базовой версии GANs минимизируется дивергенция Дженсена-Шеннона, которая является суммой двух дивергенций Кульбака-Лейблера. Они представляют из себя некоторые меры близости для пары распределений. Проблема этой метрики состоит в том, что

достаточно легко придумать два распределения, для которых эта метрика будет бесконечной, что приводит к нестабильному обучению. Данная проблема решается путем использования Wasserstein GAN, который мы рассмотрим далее.

3.2 Wasserstein GAN

Дискриминатор обучается таким образом, чтобы гарантировать, что разница в его выходе на двух выборках эквивалентна расстоянию Вассерштейна между соответствующими распределениями. Это означает, что дискриминатор на самом деле не классифицирует объекты, а его выход является непрерывной величиной, поэтому в том числе не будет использоваться функции активации на выходе с дискриминатора.

4 Архитектура

Архитектуры генератора и дискриминатора совпадают с теми, что использовались в оригинальной статье "Image-to-Image Translation with Conditional Adversarial Networks" далее приведем их краткое описание. Данные архитектуры использовались во всех экспериментах, кроме начальных, где в качестве дискриминатора использовалась 4-х слойная сверточная сеть с полно связанным слоем в конце, более детальное описание данной архитектуры можно найти ниже в соответствующем эксперименте.

5 Данные

В рамках данного домашнего задания использовались самостаятельно собранные наборы данных (взяты с kaggle), из которых затем получались парные черно-белые изображения. Первый набор данных представляет из себя 819 картинок покемонов с белым фоном. Каждая размером 256 * 256.

Ссылка на датасет: <https://www.kaggle.com/kvpratama/pokemon-images-dataset>



Рис. 2: Pokemon dataset. Example image

В качестве второго набора данных использовался датасет Anime faces, где каждая картинка имеет размер 64 * 64.

Ссылка на датасет: <https://www.kaggle.com/soumikrakshit/anime-faces>



Рис. 3: Anime Faces dataset. Example image

В силу ограниченности вычислительных ресурсов эксперименты проводились лишь на Pokemon dataset, для другого датасета будут представлены пример работы архитектуры, которая показала себя лучшим образом на Pokemon dataset.

6 Эксперименты

Подробное описание всех гиперпараметров будет представлено в следующем разделе.

6.1 Baseline

6.1.1 Гипотеза:

Возможно, не имеет смысла обучать архитектуру с использованием adversarial loss и использование одного лишь l1 loss позволит достичь приемлемого качества.

6.1.2 Описание:

В качестве baseline проводились эксперименты с использованием одного лишь генератора, архитектура которого была описана выше, данная модель будет использоваться во всех дальнейших экспериментах, для обучения использовался l1 loss.

6.1.3 Результаты:

Пример работы:

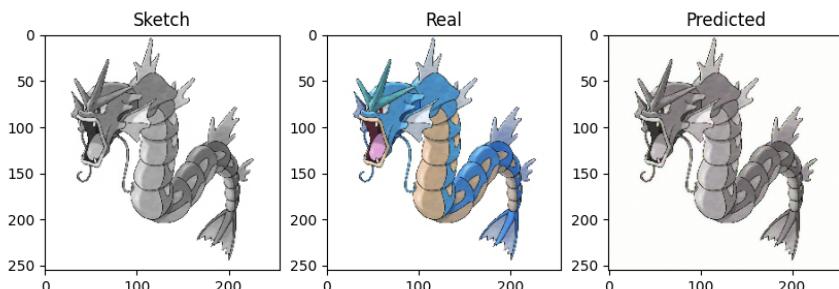


Рис. 4: Baseline. Results

Гиперпараметры:

Name	Value
device	cuda:3
epochs_num	100
g_lr	0.0003
in_channels	3
log_each	50
num_downs	7
out_channels	3
train_batch_size	1
val_batch_size	64

Рис. 5: Baseline. Config

Кривые обучения:

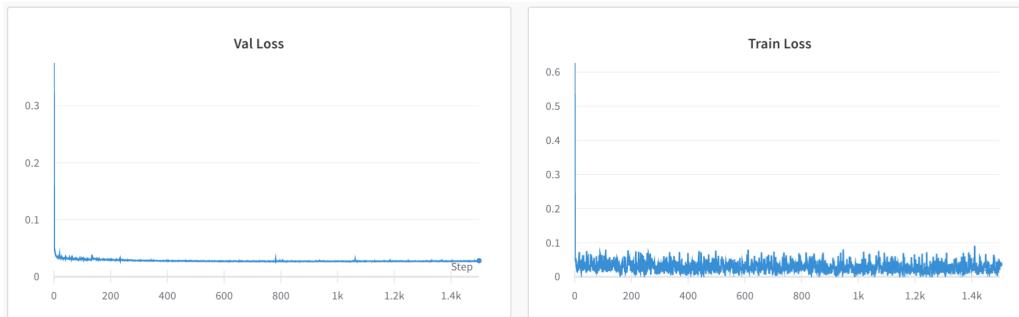


Рис. 6: Baseline. Metrics

6.1.4 Вывод:

Использование одного лишь l1 loss не позволяет решить поставленную задачу. Необходимо использование более сложных методов.

6.2 Adversarial loss

6.2.1 Гипотеза:

Использование adversarial loss позволит улучшить baseline подход и позволит получать цветные картинки

6.2.2 Описание:

В качестве дискриминатора использовалась 4-х слойная сверточная сеть

	Name	Type	# Parameters	Output Shape
●	conv1	Conv2d(3, 16, kernel_size=(4, 4), stride=(3, 3))	768, 16	1,16,85,85
●	conv2	Conv2d(16, 32, kernel_size=(4, 4), stride=(3, 3))	8192, 32	1,32,28,28
●	conv3	Conv2d(32, 64, kernel_size=(4, 4), stride=(3, 3))	32768, 64	1,64,9,9
●	conv4	Conv2d(64, 128, kernel_size=(4, 4), stride=(3, 3))	131072, 128	1,128,2,2
●	fc	Linear(in_features=512, out_features=1, bias=True)	512, 1	1,1

Рис. 7: Adversarial loss. Metrics

6.2.3 Результаты:

Пример работы:

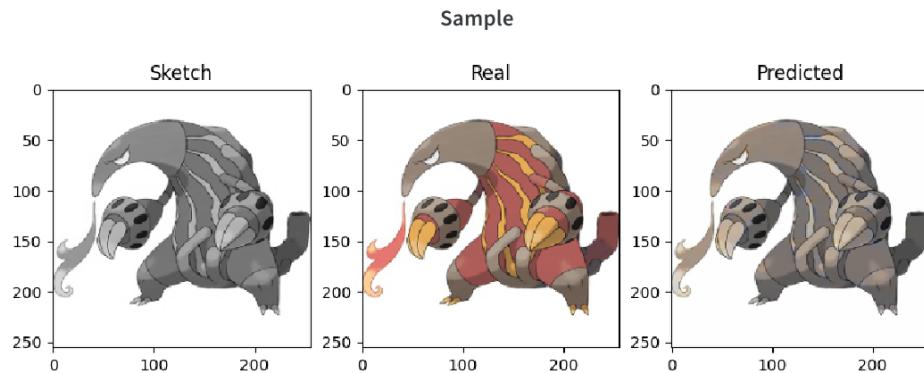


Рис. 8: Adversarial loss. Results

Гиперпараметры:

Name	Value
⚙️ d_lr	0.0003
⚙️ device	cuda:1
⚙️ epochs_num	100
⚙️ g_lr	0.0003
⚙️ in_channels	3
⚙️ lambda	10
⚙️ log_each	50
⚙️ num_downs	7
⚙️ out_channels	3
⚙️ train_batch_size	1
⚙️ val_batch_size	64

Рис. 9: Adversarial loss. Config

Кривые обучения:

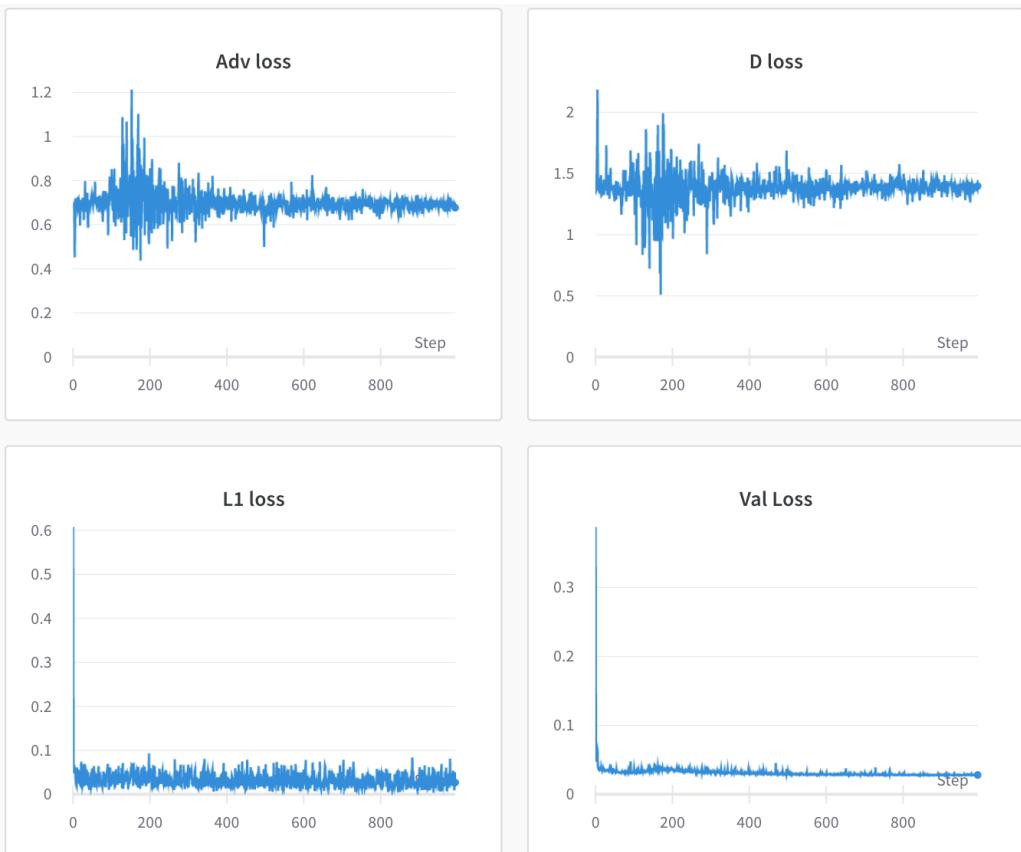


Рис. 10: Adversarial loss. Metrics

6.2.4 Вывод:

Получилось добиться появления цвета в картинках, но картинки все равно получаются почти одноцветные.

6.3 Wasserstein GAN

6.3.1 Гипотеза:

Использование Wasssertein Gan позволит получить картинки лучшего качества и позволит более стабильно обучаться.

6.3.2 Описание:

Здесь и в дальнейших экспериментах используется improved training of Wasserstein GANs, то есть вместо gradient clipping применяется gradient penalty, что в большинстве задач показывает результаты лучше. В качестве дискриминатора использовалась та же модель, что и в предыдущем эксперименте.

6.3.3 Результаты:

Пример работы:

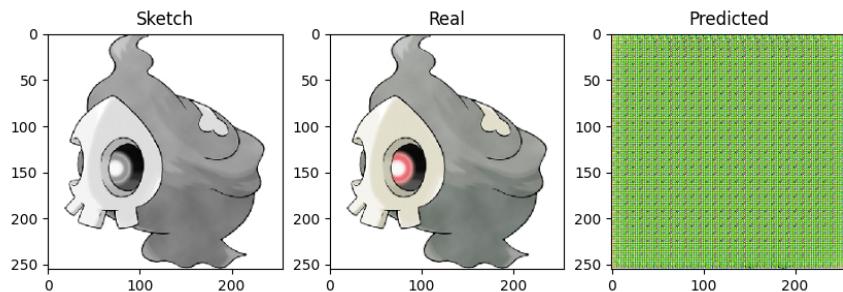


Рис. 11: WGAN. Results

Гиперпараметры:

Name	Value
⚙️ d_coef	20
⚙️ d_lr	0.00001
⚙️ device	cuda:3
⚙️ epochs_num	100
⚙️ g_lr	0.00001
⚙️ in_channels	3
⚙️ l1_weight	10
⚙️ lambda	10
⚙️ log_each	5
⚙️ num_downs	7
⚙️ out_channels	3
⚙️ train_batch_size	8
⚙️ val_batch_size	64

Рис. 12: WGAN. Config

Кривые обучения:

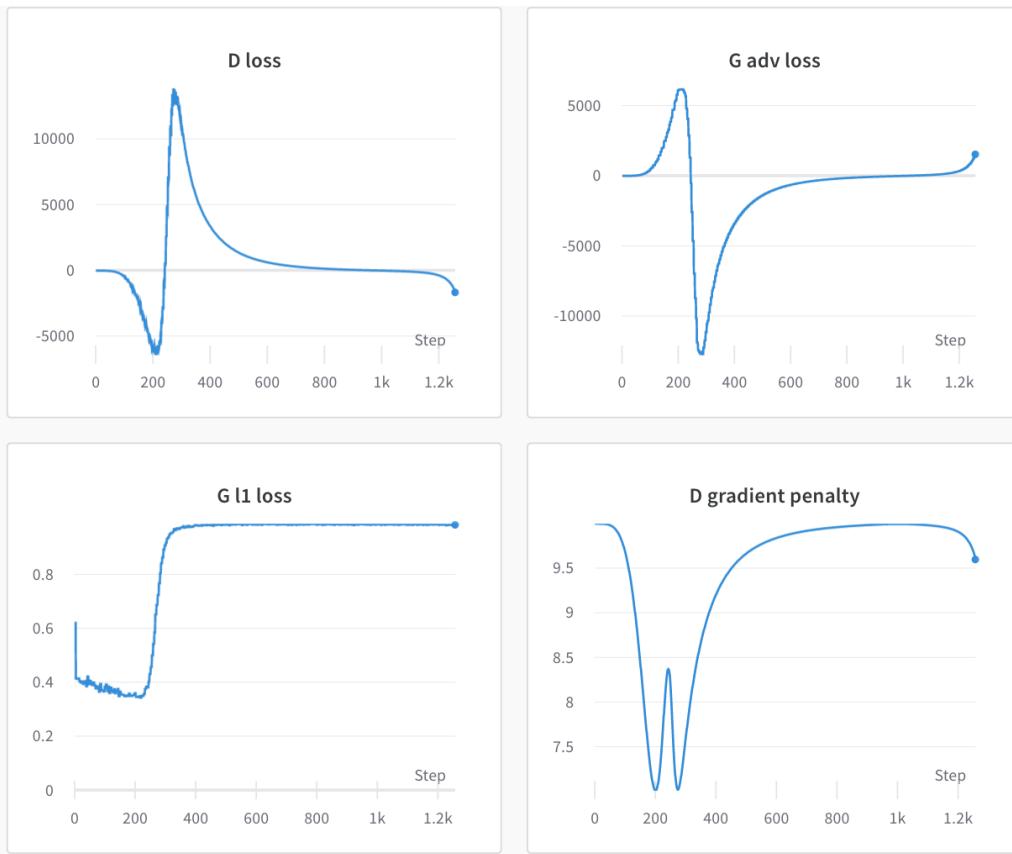


Рис. 13: WGAN. Metrics

6.3.4 Вывод:

После 200 итераций обучение разошлось, и сеть перестала генерировать что-то похожее на исходные картинки. Следует попробовать более аккуратно смешивать adversarial training и l1 loss.

6.4 Delayed adversarial training

6.4.1 Гипотеза:

Стоит более аккуратно смешивать adversarial loss и l1 loss. Для этого предлагается попробовать сначала обучать сеть несколько эпох используя только l1 loss, а затем к нему добавлять adversarial loss.

6.4.2 Описание:

В течение первых 2 эпох сеть тренировалась с использованием только l1 loss, затем на сумме лоссов. В качестве дискриминатора использовалась та же модель, что и в предыдущем эксперименте. Обучение также происходило с помощью improved WGAN, как и в дальнейших экспериментах.

6.4.3 Результаты:

Пример работы:

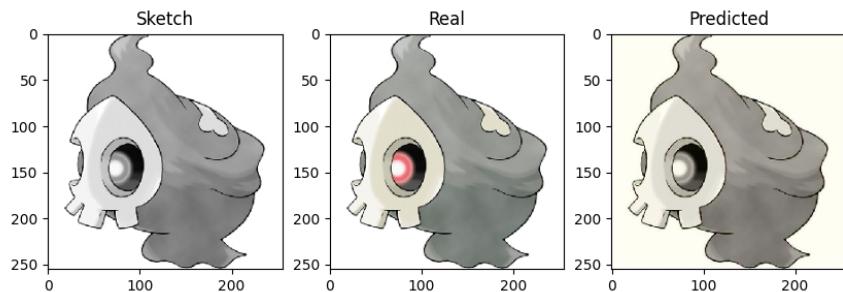


Рис. 14: Delayed adversarial training. Results

Гиперпараметры:

Name	Value
⚙️ d_coef	10
⚙️ d_lr	0.0001
⚙️ device	cuda:3
⚙️ epochs_num	100
⚙️ g_lr	0.001
⚙️ in_channels	3
⚙️ l1_weight	300000
⚙️ lambda	10
⚙️ log_each	5
⚙️ num_downs	7
⚙️ out_channels	3
⚙️ start_epoch	2
⚙️ train_batch_size	8
⚙️ val_batch_size	64

Рис. 15: Delayed adversarial training. Config

Кривые обучения:

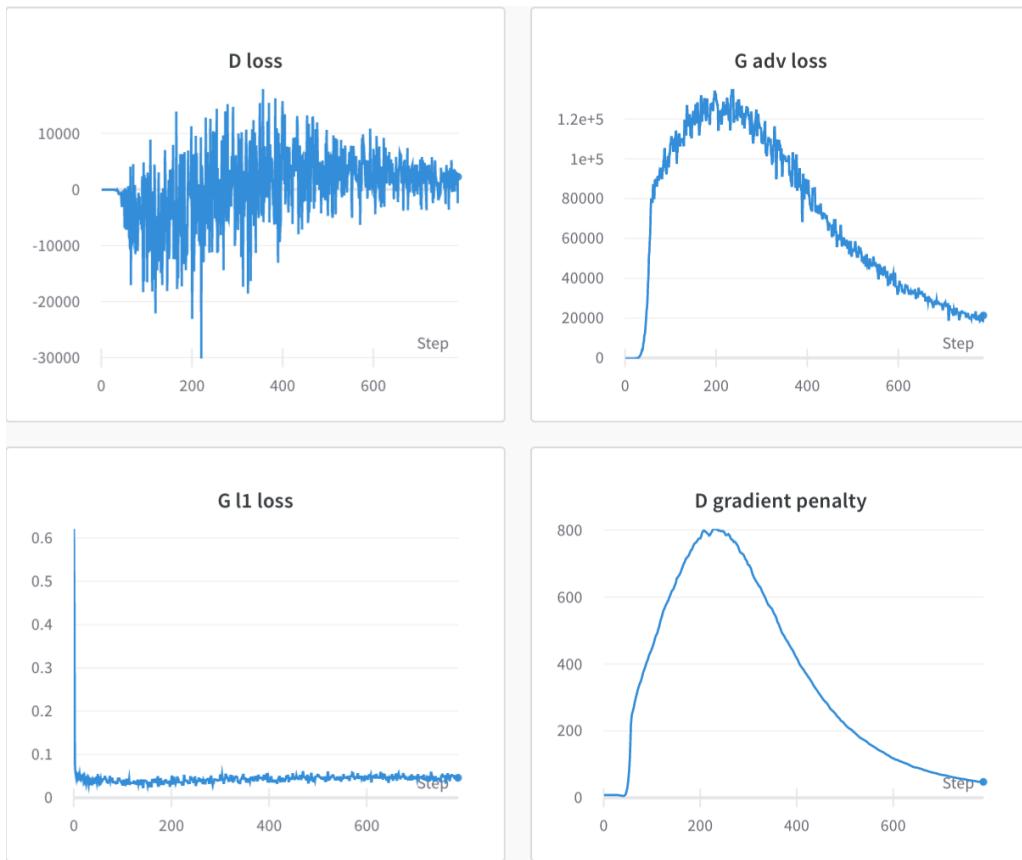


Рис. 16: Delayed adversarial training. Metrics

6.4.4 Вывод:

Используя данный подход получилось справиться с проблемой предыдущего эксперимента, во всех дальнейших экспериментах данная техника будет использоваться.

6.5 Patchgan discriminator

6.5.1 Гипотеза:

Если посмотреть на графики лоссов обучения WGAN, то можно заметить, что они становятся слишком большими по модулю, такое часто бывает при неверном выборе архитектуры модели, предлагается попробовать провести эксперимент с моделью patchgan, которая используется в оригинальной статье. Возможно, это позволит стабилизировать обучение и улучшить качество.

6.5.2 Описание:

Вместо дискриминатора, который использовался в предыдущих экспериментах, использовался patchgan discriminator из оригинальной статьи "Image-to-Image Translation with Conditional Adversarial Networks"

6.5.3 Результаты:

Пример работы:

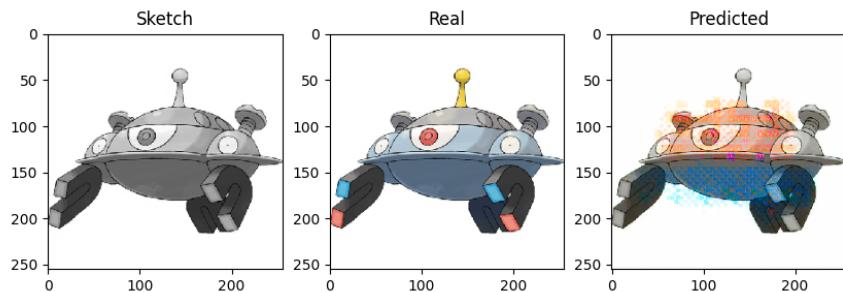


Рис. 17: Patchgan discriminator. Results

Гиперпараметры:

Name	Value
adv_d_lr	0.0001
adv_g_lr	0.001
d_coef	2
d_lr	0.0001
device	cuda:1
epochs_num	100
g_lr	0.001
in_channels	3
l1_weight	50
lambda	10
log_each	5
num_downs	7
out_channels	3
start_epoch	0
train_batch_size	8

Рис. 18: Patchgan discriminator. Config

Кривые обучения:

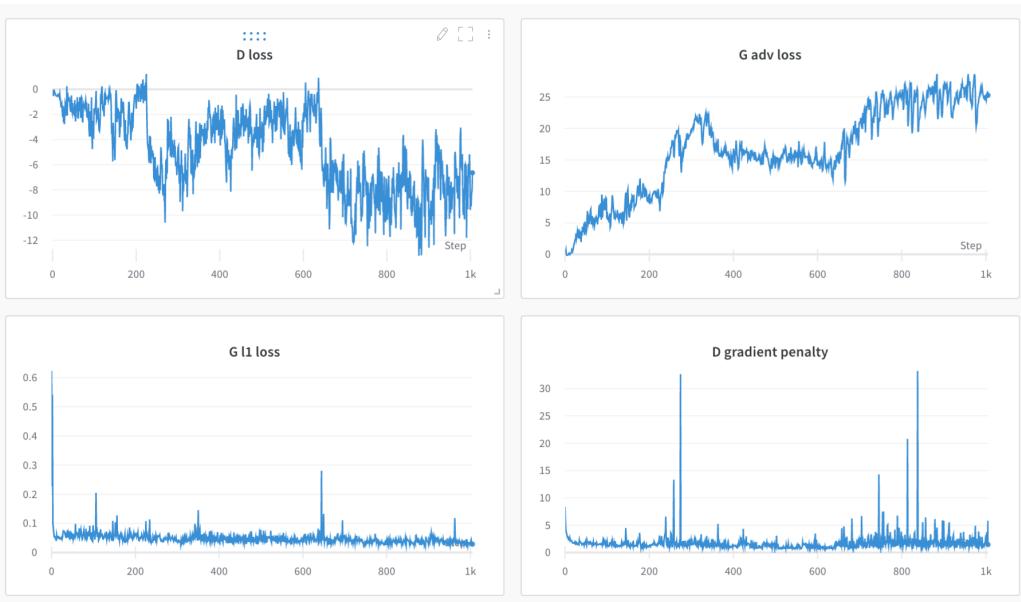


Рис. 19: Patchgan discriminator. Metrics

6.5.4 Вывод:

Использование patchgan позволило стабилизировать обучение и улучшить качество. В дальнейших экспериментах данная техника будет использоваться.

6.6 One channel input

6.6.1 Гипотеза:

В предыдущих экспериментах, несмотря на то, что использовалась черно-белая картинка на вход сети генератора, она представлялась в виде 3 каналов, возможно, использование одного лишь канала позволит более эффективно обучить сеть и улучшить качество

6.6.2 Описание:

Вместо `transforms.Grayscale(num_output_channels=3)` использовалась трансформация `transforms.Grayscale(num_output_channels=1)`

6.6.3 Результаты:

Пример работы:

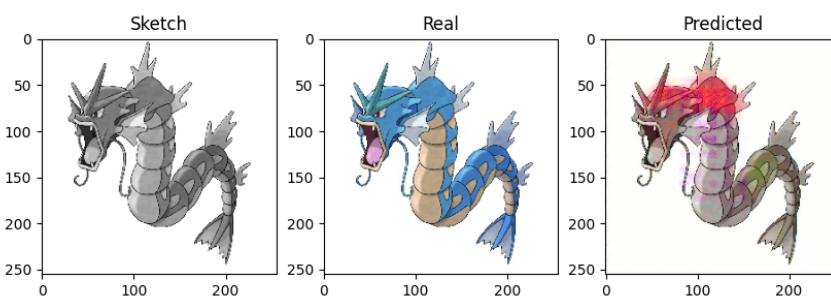


Рис. 20: One channel input. Results

Гиперпараметры:

Name	Value
⚙️ adv_d_lr	0.0001
⚙️ adv_g_lr	0.001
⚙️ d_coef	3
⚙️ d_epochs_num	2
⚙️ d_lr	0.0001
⚙️ device	cuda:3
⚙️ epochs_num	50
⚙️ g_epochs_num	2
⚙️ g_lr	0.0001
⚙️ in_channels	1
⚙️ l1_weight	50
⚙️ lambda	10
⚙️ log_each	5
⚙️ num_downs	7
⚙️ out_channels	3

Рис. 21: One channel input. Config

Кривые обучения:

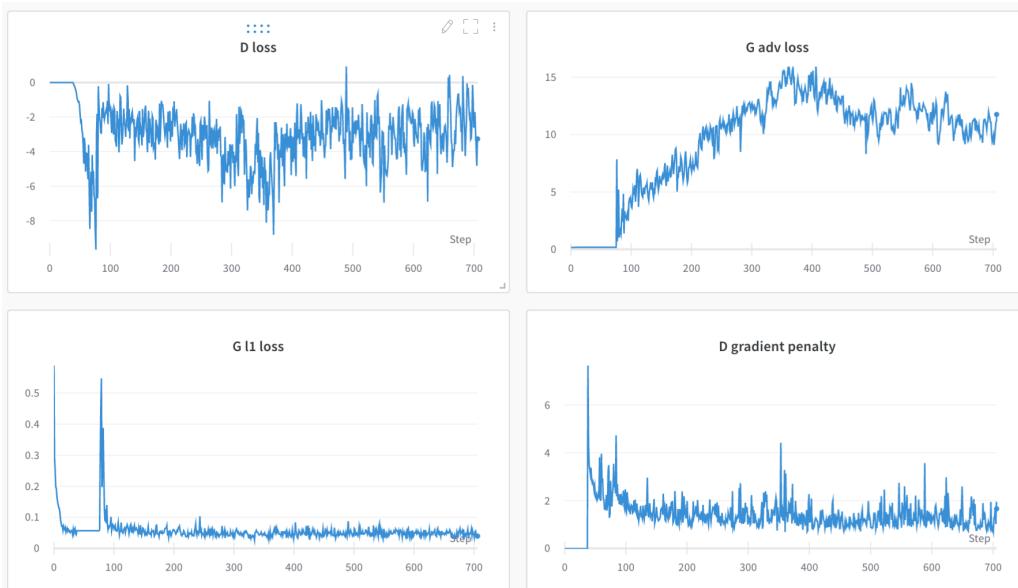


Рис. 22: One channel input. Metrics

6.6.4 Вывод:

Получилось улучшить сходимость (D_loss иллюстрирует расстояние между распределениями) и улучшить качество генерации. В дальнейших экспериментах данная техника будет использоваться.

6.7 L1 weight

6.7.1 Гипотеза:

Использование l1 weight отличного от 25 позволит получить более красочные картинки.

6.7.2 Описание:

Проводилось 5 экспериментов, которые отличались между собой только значениями l1 weight, рассматривались следующие значения: [10, 25, 50, 75, 100]. Использовался batch_size = 8 при обучении

6.7.3 Результаты:

Пример работы:

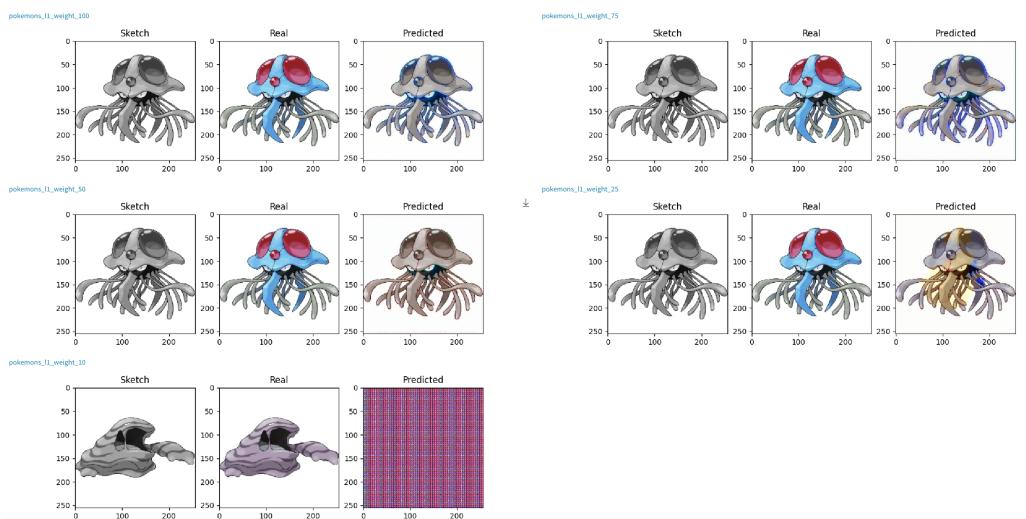


Рис. 23: L1 weight. Results (справа налево, сверху вниз по уменьшению l1 weight)

Гиперпараметры:

⚙️ adv_d_lr	0.0001
⚙️ adv_g_lr	0.001
⚙️ d_coef	3
⚙️ d_epochs_num	2
⚙️ d_lr	0.0001
⚙️ device	cuda:3
⚙️ epochs_num	20
⚙️ g_epochs_num	2
⚙️ g_lr	0.0001
⚙️ in_channels	1
⚙️ l1_weight	25
⚙️ lambda	10
⚙️ log_each	25
⚙️ num_downs	7
⚙️ out_channels	3

Рис. 24: L1 weight. Config

Кривые обучения:

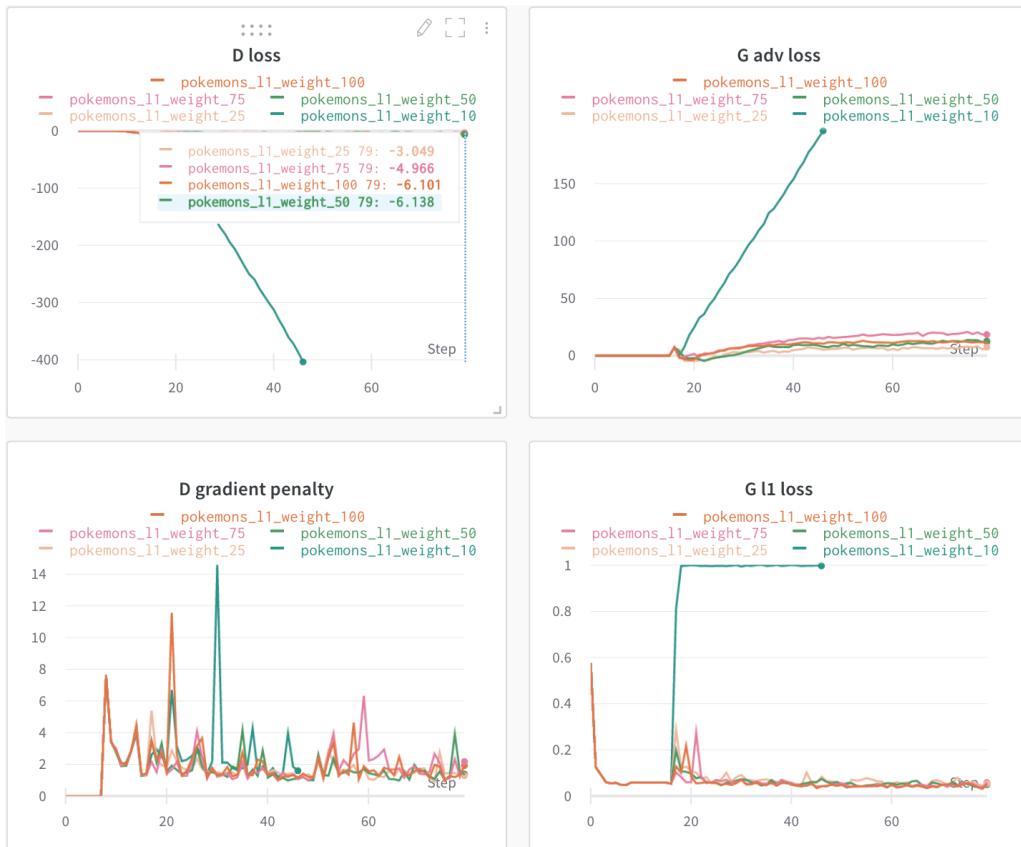


Рис. 25: L1 weight. Metrics

6.7.4 Вывод:

По качеству картинок и по D_loss видно, что лучшее качество получается при l1 weight = 25. Данное значение будет использоваться в дальнейших экспериментах.

6.8 Optimizal batch size

6.8.1 Гипотеза:

Возможно обучение с более маленьким или более большим batch size позволит улучшить качество генерируемых картинок.

6.8.2 Описание:

Проводилось 4 эксперимента, которые отличались между собой только значениями batch size, рассматривались следующие значения: [4, 8, 16, 32].

6.8.3 Результаты:

Пример работы:

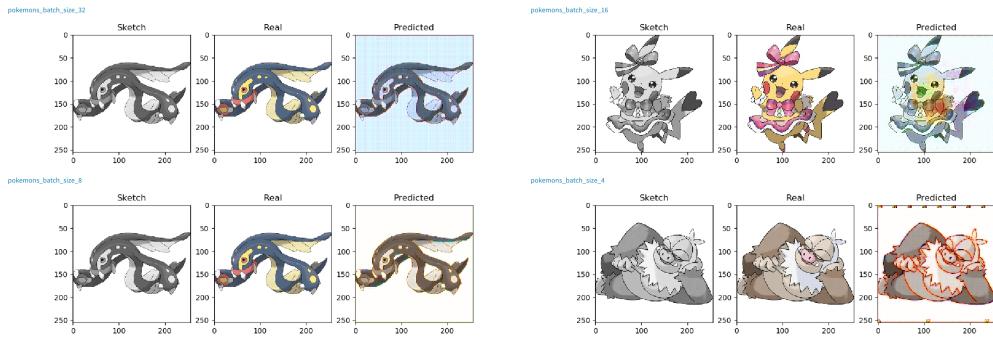


Рис. 26: Optimizal batch size. Results

Гиперпараметры использовались аналогично предыдущему эксперименту с единственным различием в batch size.

Кривые обучения:

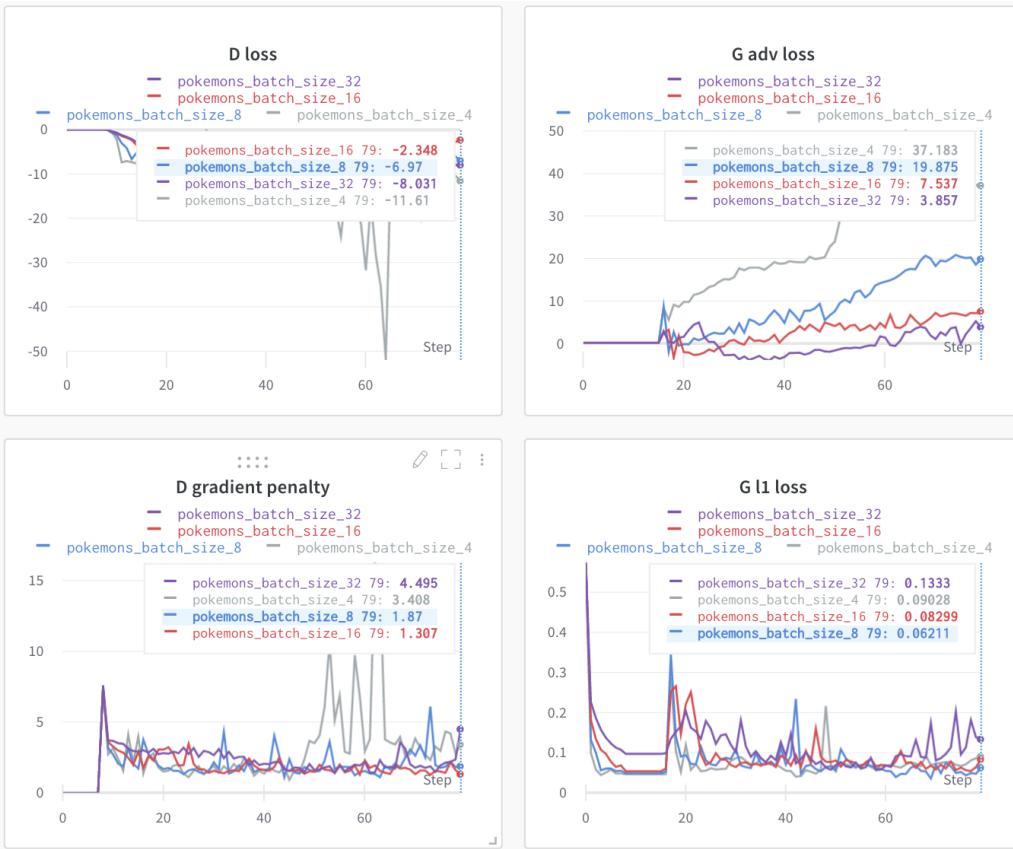


Рис. 27: Optimizational batch size. Metrics

6.8.4 Вывод:

Согласно качеству картинок лучшие всего себя показывает batch size 8, он будет использоваться в дальнейших экспериментах и в финальной модели.

6.9 Discriminator coefficient

6.9.1 Гипотеза:

Модели gan очень чувствительны к балансу между количеством итераций для обучения генератора и дискриминатора, предлагается провести эксперимент, чтобы выяснить, насколько больше следует обучать дискриминатор по сравнению с генератором.

6.9.2 Описание:

Проводилось 6 экспериментов, которые отличались между собой только значениями d_coef, рассматривались следующие значения: [1, 2, 3, 4, 5, 6]. d_coef означает во сколько раз больше итераций делает дискриминатор по сравнению с генератором.

6.9.3 Результаты:

Пример работы:

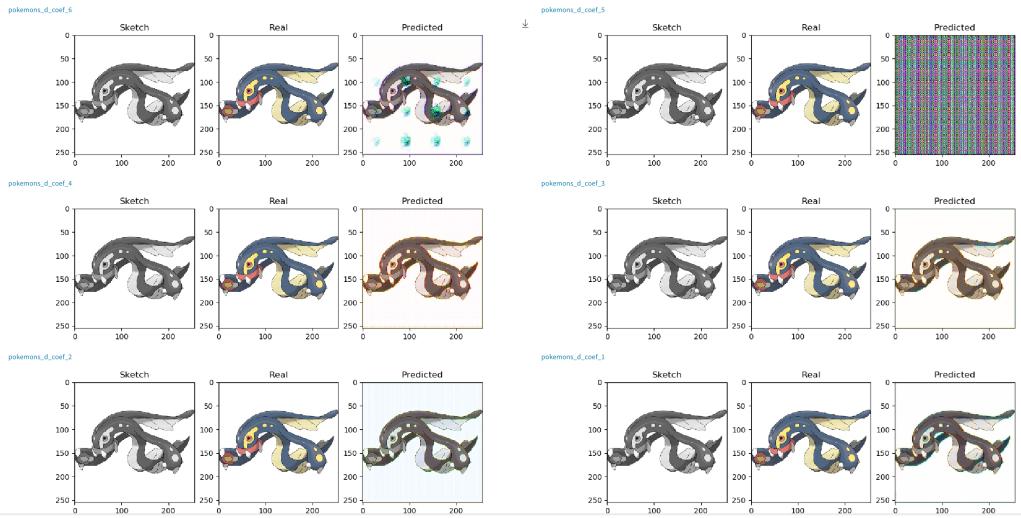


Рис. 28: Discriminator coefficient. Results

Гиперпараметры использовались аналогично предыдущему эксперименту с единственным различием в d_coef .

Кривые обучения:

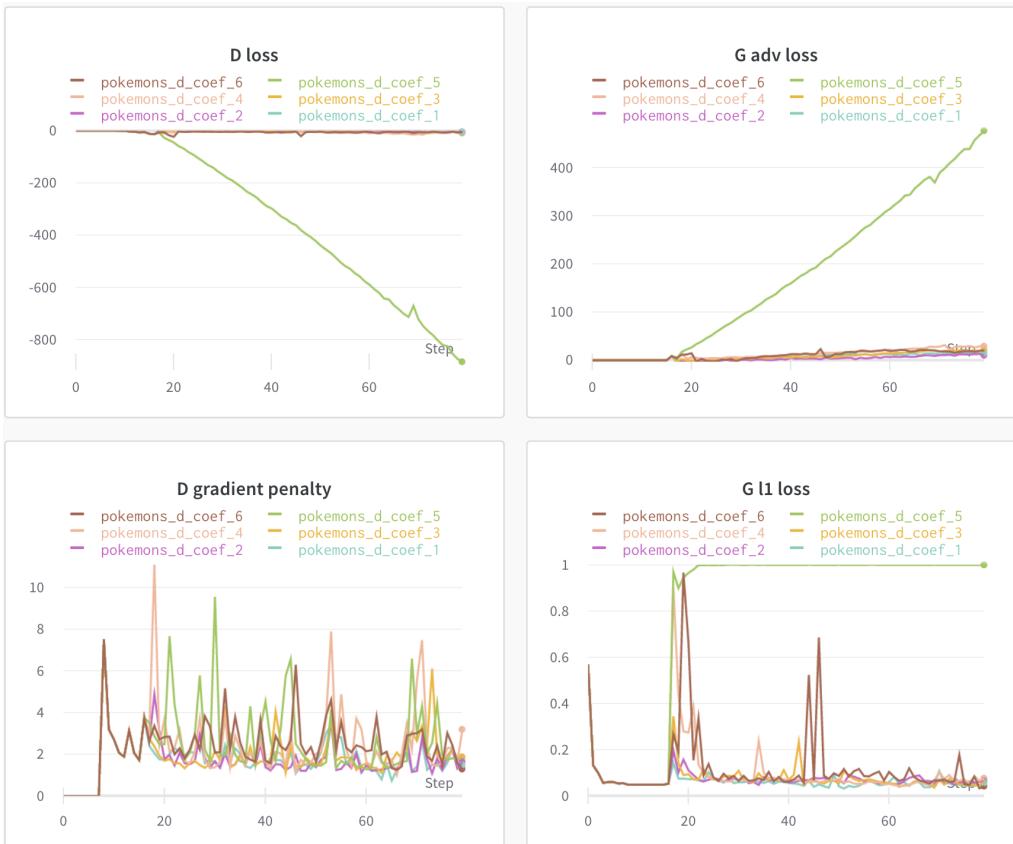


Рис. 29: Discriminator coefficient. Metrics

6.9.4 Вывод:

По качеству картинок лучше всего себя показало значение $d_coef = 3$, оно также использовалось в финальной архитектуре.

7 Гиперпараметры

- adv_d_lr - learning rate, который используется для обучения дискриминатора при одновременном обучении с генератором
 - adv_g_lr - learning rate, который используется для обучения генератора при одновременном обучении с дискриминатором
 - d_coef - во сколько раз больше итераций делает дискриминатор во время обучения
 - d_epochs_num - количество эпох, которое учится дискриминатор перед совместной тренировкой с генератором
 - d_lr - learning rate, который используется для обучения дискриминатора при одиночной тренировке
 - device - cuda device
 - epochs_num - количество эпох обучения
 - g_epochs_num - количество эпох, которое учится генератор перед совместной тренировкой с дискриминатором (сперва учится генератор g_epochs_num, затем дискриминатор d_epochs_num, затем совместно)
 - g_lr - learning rate, который используется для обучения генератора при одиночной тренировке
 - in_channels - количество входных каналов
 - l1_weight - с каким весом учитывать l1 loss в финальном loss ($l1_weight * l1\ loss + adv\ loss$)
 - lambda - gradient penalty coefficient
 - log_each - частота логирования метрик
 - num_downs - U-net number of downs
 - out_channels - размерность выходного канала картинки (3)
 - train_batch_size - batch size при обучении
 - val_batch_size - batch size при валидации
- filters_num = 64, kernel_size = 4, stride = 2, padding = 1 не указываются в конфигах, так как совпадают во всех экспериментах

8 Примеры работы

8.1 Pokemon dataset

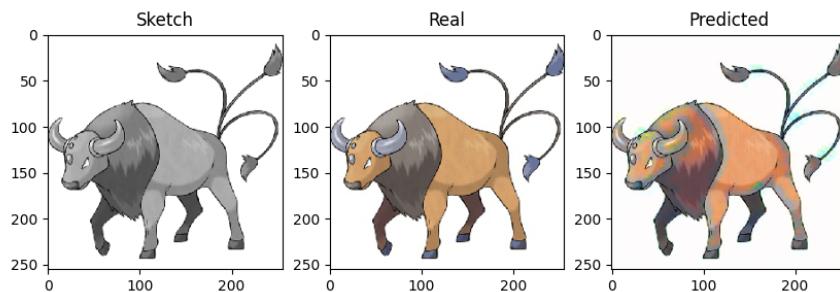


Рис. 30: Pokemon dataset. Example

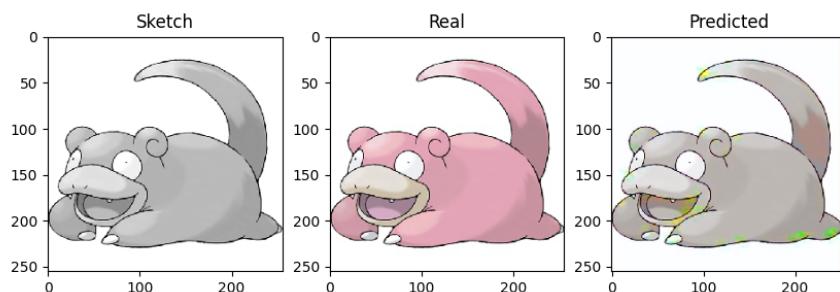


Рис. 31: Pokemon dataset. Example

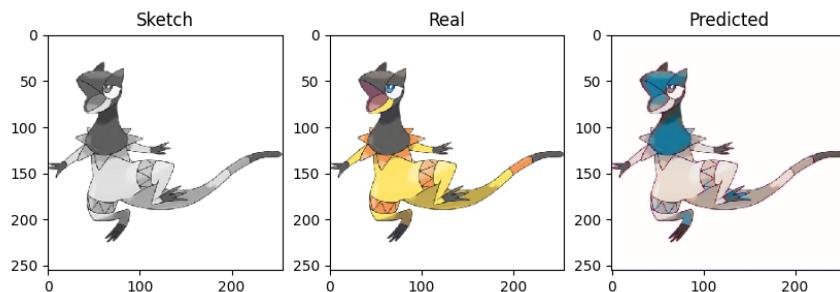


Рис. 32: Pokemon dataset. Example

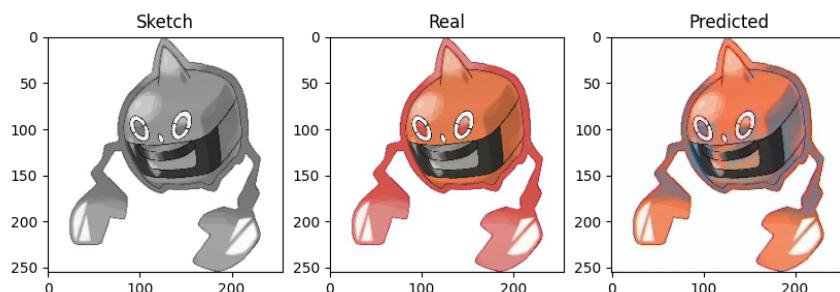


Рис. 33: Pokemon dataset. Example

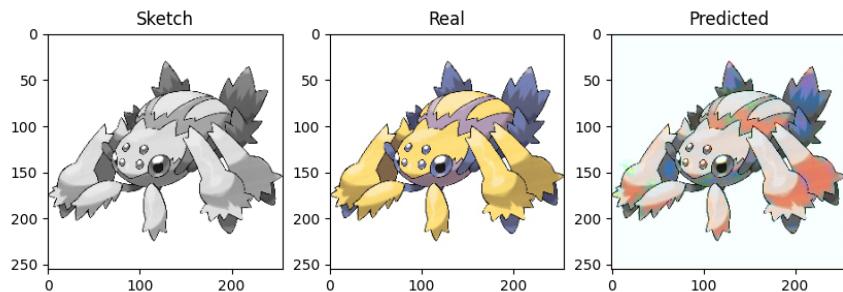


Рис. 34: Pokemon dataset. Example

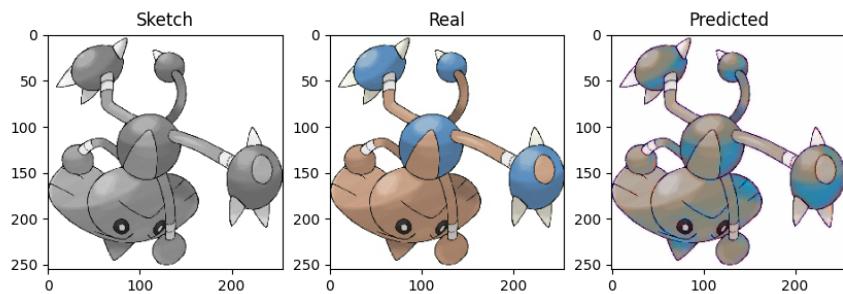


Рис. 35: Pokemon dataset. Example

8.2 Anime faces

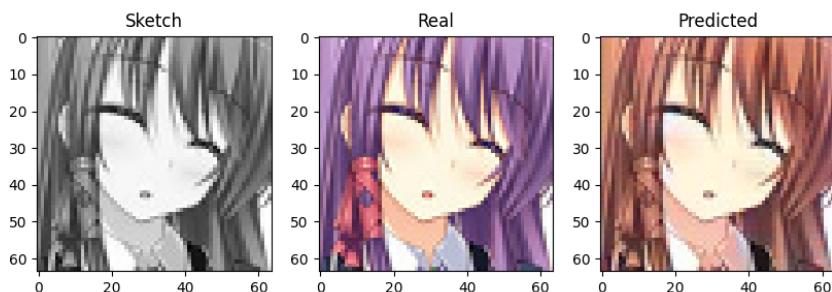


Рис. 36: Anime dataset. Example

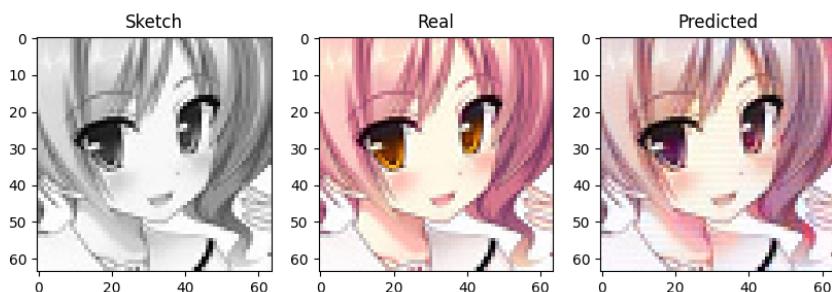


Рис. 37: Anime dataset. Example

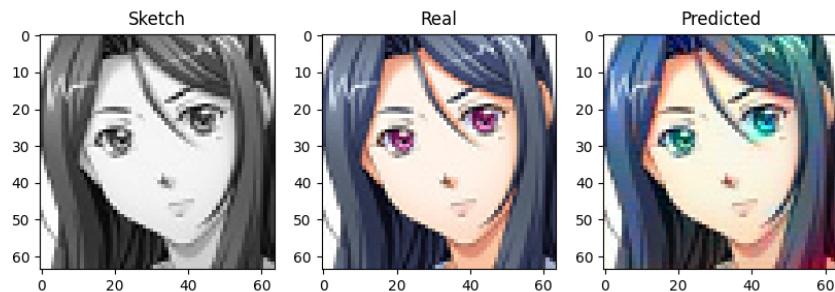


Рис. 38: Anime dataset. Example

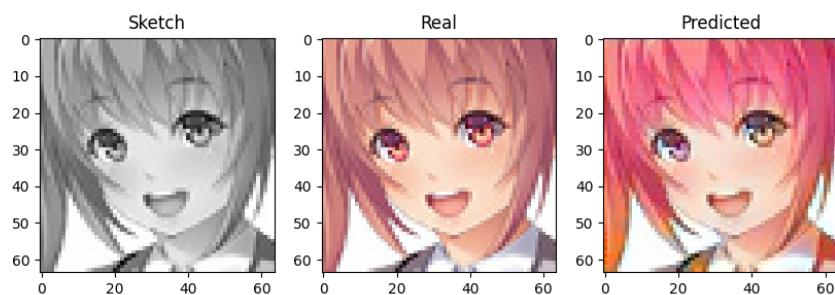


Рис. 39: Anime dataset. Example

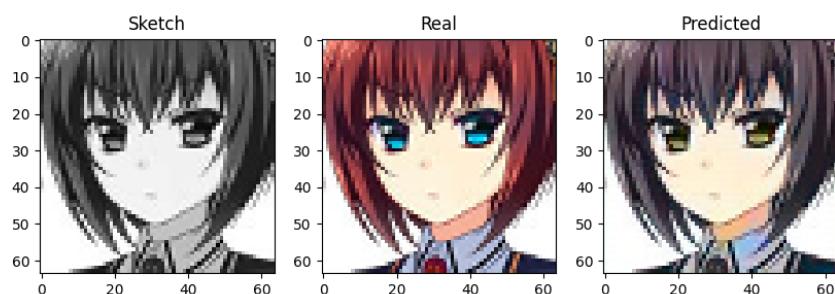


Рис. 40: Anime dataset. Example

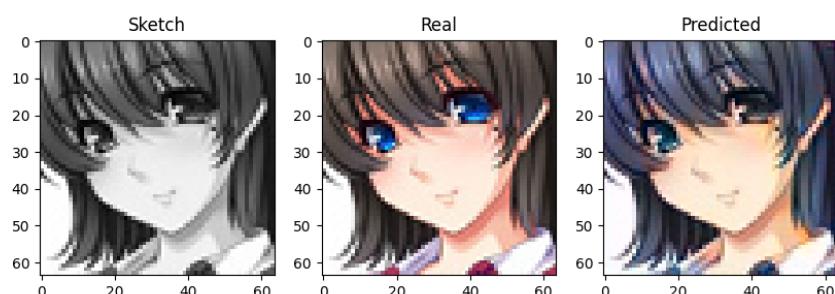


Рис. 41: Anime dataset. Example