Содержание

1 Компоненты свёрточной нейрнной сети **2**
1.1 Свёртка 2
1.2 Свёрточный слой 3
1.3 Pooling слой 3

2 Свёрточная нейронная сеть **4**

3 Улчшения качества свёрточных нейросетей **5**
3.1 Аугументация 5
3.2 Inception модуль 6

4 Выводы **7**

5 GAN — Порождающие состязательные сети **8**
5.1 Постановка задачи 8
5.2 Алгоритм обучения 10
5.3 Проблемы обучения GAN 11

6 CGAN — Условные порождающие состязательные сети **12**
6.1 DCGAN 13

7 StackGAN **13**
7.1 Stage I 14
7.2 Stage II 14

8 LAPGAN **15**

9 Сегментация **15**

1 Компоненты свёрточной нейрнной сети

Определение 1. Изображение — тензор $M \in \mathbb{R}^{m \times n \times d}$, m — ширина изображения, n — длина. Чаще всего $d = 3$ (3 канала — Red, Green, Blue).

Учитывая, что обычно рассматривают трёхканальные или четырёхканальные изображения, можно под словом "тензор"понимать трёхмерный или четырёхмерный массив.

Пусть $X \in \mathbb{R}^{m \times n \times d}$ — случайная величина "изображение".

Рассматриваются следующие задачи для изображений:

- Классификация $f : X \to \{1, \dots, K\}$, K — число классов
- Сегментация (1 объект) $f : X \to Y$, $Y \in [0,1]^{m \times n}$, $Y_{ij} = \mathsf{P}(X_{ij} \in \text{segment})$, где segment — множество точек изображения (пикселей), принадлежащих объекту. Понятно, что можно обобщить на сегментацию нескольких классов объектов.

1.1 Свёртка

Определение 2. Пусть $f, g : \mathbb{R}^n \to \mathbb{R}$, интегрируемые на $\mathbb{R}^n$. *Свёрткой* будем называть функцию $f*g : \mathbb{R}^n \to \mathbb{R}$, равную $(f*g)(x) = \int\limits_{\mathbb{R}^n} f(y)g(x-y)dy$

Далее под свёрткой будем понимать следующее определение.

Определение 3. Рассмотрим $M \in \mathbb{R}^{m \times n}$ — изображение по одному из каналов, $K \in \mathbb{R}^{k \times l}$ — ядро, *свёрткой изображения относительно отображения* $h : \mathbb{R}^{m \times n} \times \mathbb{R}^{k \times l} \to \mathbb{R}$ *называется функция* $*$, рассмотрим, $(M*K)(i,j) = \sum\limits_{a=1}^{k} \sum\limits_{b=1}^{l} M_{i+a,j+b} K_{ab}$. $(M * K) \in \mathbb{R}^{(m-k+1) \times (n-l+1)}$

Для того чтобы размер изображения после свёртки был равен исходному изображению можно как-нибудь заполнить края изображения (padding). Идея аналогична скользящему среднему с синхронизацией в центральной точке. В основном используются следующие способы padding:

- zeros — заполнение нулями.
- reflect — "отражение например, вектор $(1,2,3)$ при padding размера 2 отобразится в вектор $(3,2,1,2,3,2,1)$.
- replicate — заполнение крайними значениями.
- circular — круговое заполнение, вектор $(1,2,3)$ преобразуется в $(2,3,1,2,3,1,2)$.

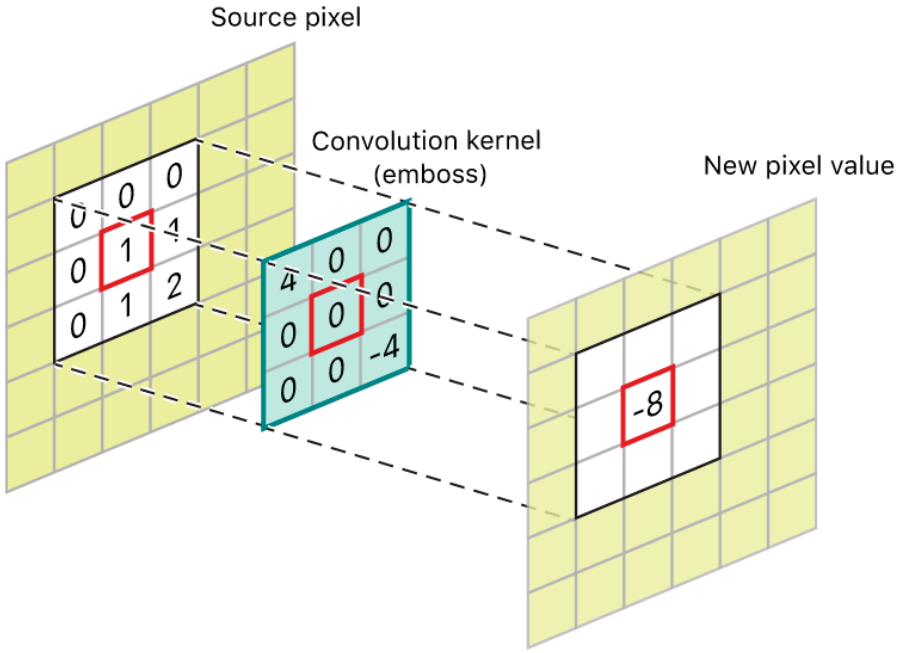

Рис. 1: Операция свёртки

1.2 Свёрточный слой

Свёрточный слой задаётся следующими параметрами:

- Размер ядер $k \times l$, $k \leqslant m$, $l \leqslant n$, $r \in \mathbb{N}$ число ядер
- Способ заполнения краёв (padding)
- Размер заполнения краёв $P \in \mathbb{N}_0$
- Величина сдвига ядра (stride) $S \in R^2$. Например, для $S = (1, 1)$ получаем операцию, похожую на операцию вложения в 2D-SSA.

1.3 Pooling слой

Свёрточная нейросеть помимо свёрточных слоёв состоит из pooling слоёв

Определение 4. Рассмотрим одноканальное изображение (матрицу) M размера $m \times n$. Выберем p и q, кратные m и n соответственно. Разобьём матрицу на дизъюнктные подматрицы размера $k \times l$ $P_{ij} = \{M_{(i-1)p+k,(j-1)q+l}\}_{k=1,l=1}^{p,q}$, $i \in 1 : \frac{m}{p}$, $j \in 1 : \frac{n}{q}$. *Операция pooling* применяет к каждой матрице P_{ij} некоторую функцию f, в результате получается матрица F, состоящая из элементов $F(i, j) = f(P_{ij})$.

Чаще всего используют max pooling, average pooling и sum pooling. Pooling слой применяется для уменьшения размерности изображения. В случае от-

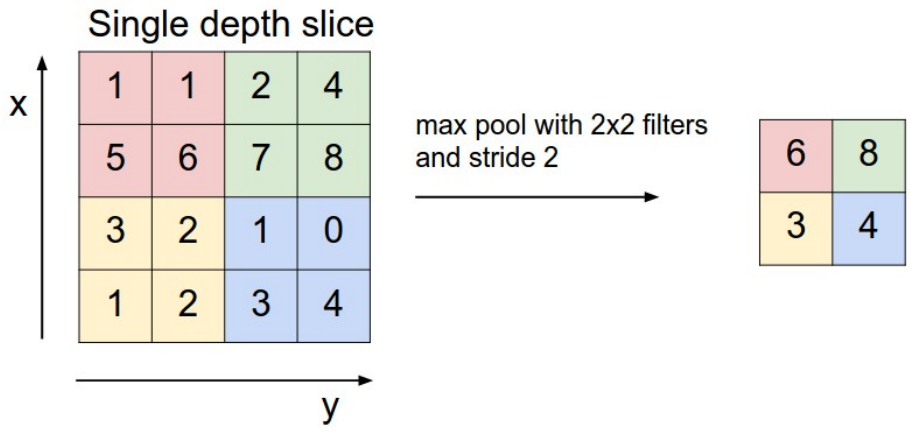

Рис. 2: Пример max-pooling

сутствия max-pooling слоёв применим алгоритм обратного распространения ошибки. Если присутствуют max-pooling слои, то градиенты пробрасываются в ту клетку, на которой достигается максимум, остальные градиенты равны нулю.

В рис. 1.3 ненулевые градиенты будут в точках $(2,2),(2,4),(3,1),(4,4)$.

На практике не встречается использование min-pooling потому что в случае его использования может произойти затухание градиентов.

2 Свёрточная нейронная сеть

В зависимости от постановки задачи после применения свёрточных и pooling слоёв и применения функций активации могут следовать полносвязные или свёрточные слои. Обычно после применения свёртки следует поэлементное применение функции ReLU, равной $\mathrm{ReLU}(x) = \max(0, x)$ для того, чтобы убрать отрицательные элементы из результатов применения свёрточных слоёв.

Каждый элемент изображения M_{ijk} показывает интенсивность выбранного пикселя соответствующей координаты и соответствующего канала, в таком случае трудно трактовать отрицательные значения в изображении.

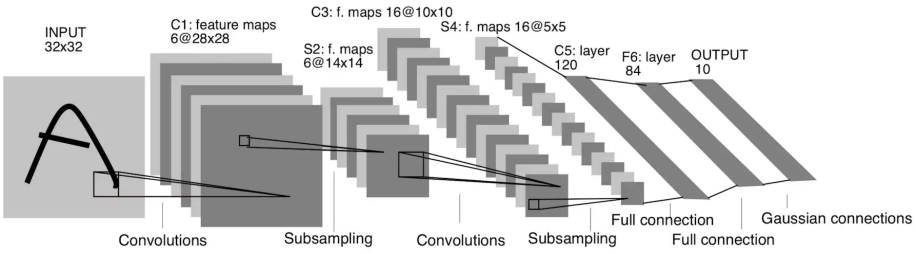

Рис. 3: Архитектура сети LeNet

Рассмотрим пример сети LeNet. На первом шаге применяется 6 свёрток размером 5×5, затем max-pooling размером 2×2, потом 16 свёрток 5×5 и max-pooling 2×2, после этого 16 полученных матриц размером 5×5 вытягиваются в один вектор, после которого следует два полносвязанных слоя, и на последнем шаге применяется функция активации softmax. В результате получаем вероятность принадлежности одному из десяти классов для каждого класса.

3 Улчшения качества свёрточных нейросетей

3.1 Аугументация

Одной из проблем работы с изображениями является малое количество размеченных изображений. Обычно изображения размечаются вручную. Для размножения данных используются аугументации.

Определение 5. Аугументация (Augumentation) — увеличение объёма тренировочной выборки с помощью различных афинных преобразований изображений: зеркальное отражение, поворот, сдвиг, изменение масштаба.

Также аугументации позволяют улучшить качество предсказания на тестовой выборки. Например, наша задача — определение дорожных знаков по фотографии. Обучающая выборка состоит из дорожных знаков из справочника (сфотографированы "анфас"). Если обучить модель на выборке без аугументаций, то при рассмотрении реальной фотографии дорожных знаков модель может давать плохие результаты потому что как правило встречаются фотографии с некоторыми дефектами в сравнении с изображениями из обучающей выборки: повороты, размытия, сдвиги. Если рассмотреть аугументированную выборке, то в ней будет больше изображений, похожих на реальные и новая модель сможет предсказывать с большей точностью чем модель, обученная на выборке без аугументаций.

Рис. 4: Пример аугументации изображений

3.2 Inception модуль

При рассмотрении свёрточного слоя непонятно ядро какого размера следует рассматривать. В качестве решения этой проблемы предлагается внедрение inception модуля в архитектуру сети. Рассмотрим несколько разных комбинаций свёрточных и pooling слоёв, у которых размерности входов и выходов по первым двум координатам (ширине и высоте тензора) равны, пройдём по этим слоям, в результате получим множество тензоров $\{T_i\}_{i=1}^{r}$, где w — число рассматриваемых комбинаций свёрток pooling. Каждый тензор T_i имеет размерность $u \times v \times w_i$. На последнем шаге тензоры T_i соединяются вдоль третьей размерности. Рассмотрим некоторый inception модуль на рис. 3.2. Он состоит из четырёх подмодулей — различных комбинаций свёрток и/или pooling. Первый подмодуль — свёртка 1×1, второй подмодуль — свёртка 1×1 и 3×3, третий состоит из последовательных свёрток 1×1 и 5×5, четвёртый из max-pooling 3×3 и свёртки 1×1. В итоге получаем четыре разных тензора, которые затем соединяем вдоль третьей размерности.

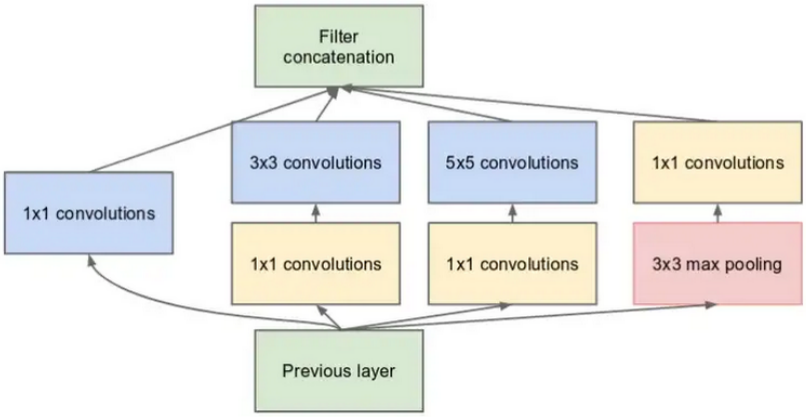

Рис. 5: Пример inception модуля

4 Выводы

Проблемы:

- Необходимость разметки данных для обучения
- Большое количество параметром, следовательно долгое обучение, даже на GPU

Решения:

- Использование размеченных библиотек изображений: ImageNet (14M изображений, 1000 категорий), OpenImages (9M изображений, 60K меток, 20K категория)
- Использование модели, обученной на размеченной библиотеке изображений известной архитектуры, например Alexnet, vgg net, Resnet.

  Обычно новые архитектуры публикуются в результате достижения рекордной точности на каком-нибудь соревновании. Например, архитектура vgg 16 показала точность 0.927 на датасете ImageNet на соревновании ImageNet Large Scale Visual Recognition Challenge 2014 и после была опубликована.

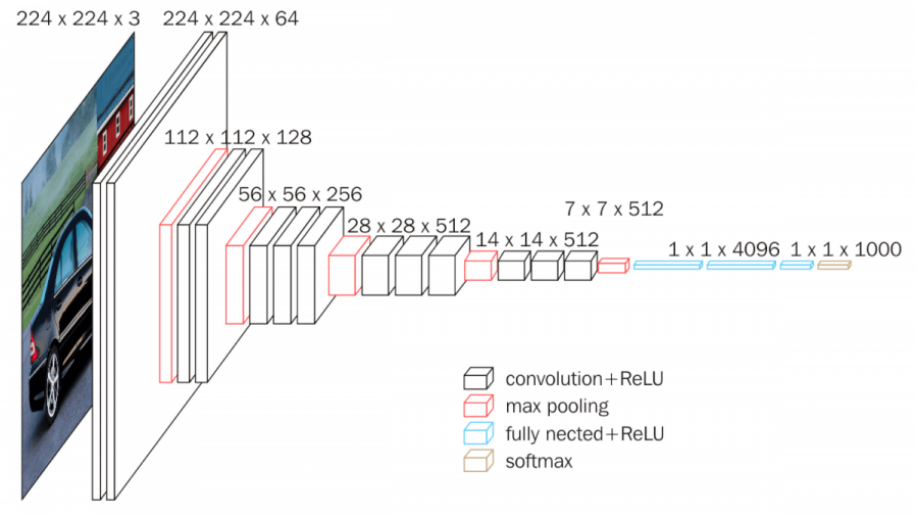

Рис. 6: Архитектура сети vgg 16

5 GAN — Порождающие состязательные сети

Порождающие состязательные сети — алгоритм машинного обучения, входящий в семейство порождающих моделей и построенный на комбинации из двух нейронных сетей: генеративная модель G, которая строит приближение распределения данных, и дискриминативная модель D, оценивающая вероятность, что образец пришел из тренировочных данных, а не сгенерированных моделью G. Обучение для модели G заключается в максимизации вероятности ошибки дискрминатора D.

Такие сети используются для следующих целей:

- Увеличение разрешения изображений.
- Преобразование текста в изображение.
- Раскрашивание изображений.
- Генерация большего количества данных и заполнение пробелов в них.

5.1 Постановка задачи

Пусть у нас имеется d-мерный набор данных из n индивидов — $\left\{x_i \in \mathbb{R}^d\right\}_{i=1}^n$; генератор G, такой что $G : z \to x$, $G(z) = x$, где $z \sim p_z(z)$ — случайный шум из нормального распределения.

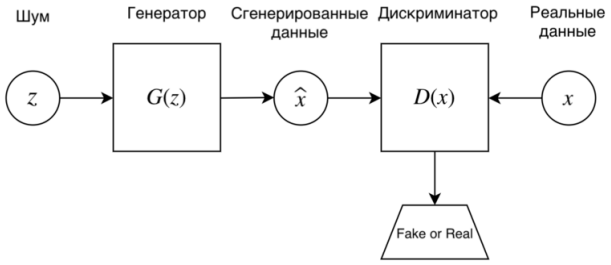

Рис. 7: Схема GAN

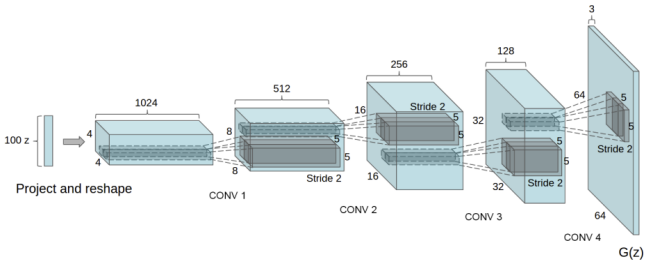

Рис. 8: Генератор

Из шума, создаются данные x (например изображение). Цель генератора — сгенерировать такое изображение x, чтобы оно было максимально похоже на изображение из исходных данных x.

Помимо генератора, в модели GAN существует дискриминатор D. По сути, это простой двоичный классификатор с сигмоидальной функцией активации, который в итоге отображает входное изображение в интервал $[0, 1]$ следующим образом.

$$D : x \to [0, 1]$$

Здесь мы предполагаем, что значение дискриминатора 1 будет означать, что x происходит из реального набора изображений, в то время как значение 0 представляет сгенерированное поддельное изображение.

$$D(x) = \begin{cases} 1, & \text{if } x \text{ is real} \\ 0, & \text{if } x \text{ is generated (fake)} \end{cases}$$

$p_{data}(x)$ — Распределение реальных данных
$p_z(z)$ — Распределение сгенерированных данных
Задача:

$$\min_G \max_D V(D, G) = \underset{x \sim p_{data}}{E} [\log D(x)] + \underset{z \sim p_z}{E} [\log (1 - D(G(z)))]$$

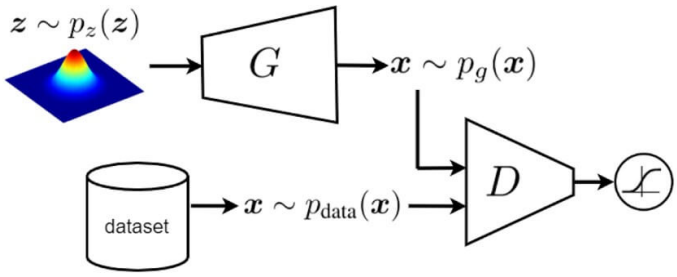

Рис. 9: GAN

Здесь необходимо оптимизировать функцию $V(D, G)$. Она состоит из двух членов. Первый связан с реальными изображениями x, которые имеют распределение $p_{data}(x)$.

Второй связан с изображениями, которые генерируются с помощью источника шума z.

5.2 Алгоритм обучения

1. Получаем мини-батч $z_1, ..., z_m$ из распределения p_z,
2. Получаем мини-батч $x_1, ..., x_m$ из распределения p_{data}
3. Обновляем дискриминатор в сторону возрастания его градиента

$$d_w \leftarrow \nabla_{\gamma_d} \frac{1}{m} \sum_{t=1}^{m} [\log D(x_t)] + [\log (1 - D (G(z_t)))]$$

4. Повторяем шаги 1-3 k раз.
5. Получаем мини-батч $z_1, ..., z_m$ из распределения p_z
6. Обновляем генератор в сторону убывания его градиента

$$g_w \leftarrow \nabla_{\gamma_g} \frac{1}{m} \sum_{t=1}^{m} [\log (1 - D (G(z_t)))]$$

На практике не всегда удобно использовать уравнение описанной выше. В начале обучения, когда G плохо настроен дискриминатор D может не учитывать объекты, с высокой уверенностью в классификации, так как они сильно отличаются от тренировочного сета, в таком случае $\log (1 - D (G(z)))$ стагнирует. Чтобы избежать этого, можно вместо минимизации $\log (1 - D (G(z)))$ максимизировать $\log (D (G(z)))$. На рисунке 3 представлена зависимость получаемого изображения от итерации обучения. На рисунке 10 представлена зависимость получаемого изображения от итерации обучения.

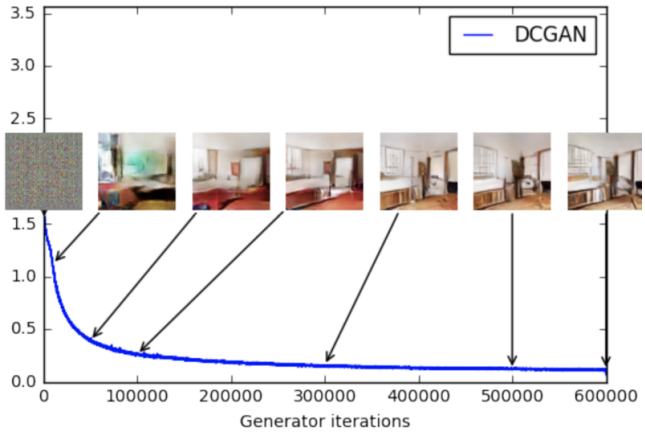

Рис. 10: GAN

5.3 Проблемы обучения GAN

- Генератор выдает ограниченное количество разных образцов.
- Параметры модели дестабилизируются и не сходятся.
- Дискриминатор становится слишком сильным, а градиент генератора исчезает и обучение не происходит.
- Выявление корреляции в признаках, не связанных (слабо связанных) в реальном мире.
- Высокая чувствительность к гиперпараметрам.

Существуют практические советы, которые могут помочь при обучении GAN'ов. Основными из них являются:

- Нормализация данных. Все признаки в диапазоне $[-1, 1]$;
- Замена функции ошибки для G с $min \log(1 - D)$ на $max \log D$, потому что исходный вариант имеет маленький градиент на раннем этапе обучения и большой градиент при сходимости, а предложенный наоборот;
- Сэмплирование из многомерного нормального распределения вместо равномерного;
- Использовать нормализационные слои (например, batch normalization или layer normalization) в G и D;
- Использовать метки для данных, если они имеются, то есть обучать дискриминатор еще и классифицировать образцы.

6 CGAN — Условные порождающие состязательные сети

Условные порождающие состязательные сети — это модифицированная версия алгоритма GAN, которая может быть сконструирована при помощи передачи дополнительных данных y, являющихся условием для генератора и дискриминатора. y может быть любой дополнительной информацией, например, меткой класса, изображением или данными из других моделей, что может позволить контролировать процесс генерации данных. Например, можно подавать параметр y, как условие на класс для генерации чисел, похожих на MNIST. Создание таких картинок, в случае передачи картинки в качетсве y является задачей трансляции изображений. Пример работы CGAN на датасете MNIST с метками классов представленных в виде one-hot векторов (рис. 11).

В дискриминаторе x и y представлены как входные параметры. В таком случае задача оптимизации будет выглядеть следующим образом:

$$\min_G \max_D V(D,G) = \underset{x\sim p_{data}}{E}[\log D(x|y)] + \underset{z\sim p_z}{E}[\log(1 - D(G(z|y)))]$$

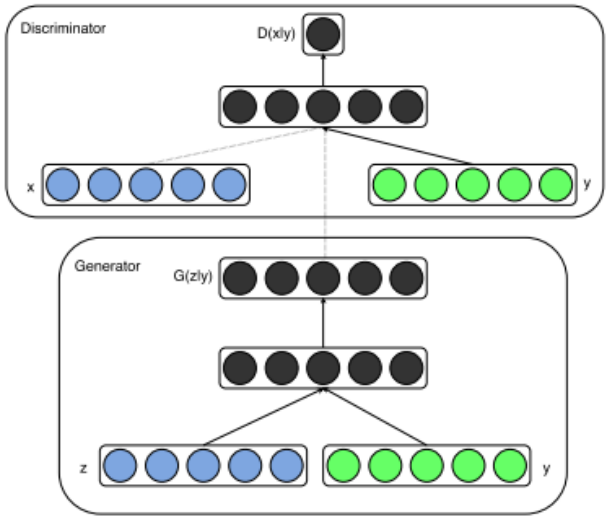

Рис. 11: CGAN

6.1 DCGAN

DCGAN — модификация алгоритма GAN, в основе которых лежат сверточные нейронные сети (CNN). Задача поиска удобного представления признаков на больших объемах не размеченных данных является одной из наибольнее активных сфер исследований, в частности представление изображений и видио. Одним из удобных способов поиска представлений может быть DCGAN. Использование сверточных нейронных сетей напрямую не давало хороших результатов, поэтому было внесены ограничения на слои сверток. Эти ограничения и лежат в основе DCGAN:

- Замена всех пулинговых слоев на страйдинговые свертки (strided convolutions) в дискриминаторе и частично-страйдинговые свертки (fractional-strided-convolutions) в генераторе, что позволяет сетям находить подходящие понижения и повышения размерностей;
- Использование батчинговой нормализации для генератора и дискриминатора, то есть нормализация входа так, чтобы среднее значения было равно нулю и дисперсия была равна единице. Не стоит использовать батч-нормализация для выходного слоя генератора и входного дискриминатор.
- Удаление всех полносвязных скрытых уровней для более глубоких архитектур;
- Использование ReLU в качестве функции активации в генераторе для всех слоев, кроме последнего, где используется tanh;
- Использование LeakyReLU в качестве функции активации в дискриминаторе для всех слоев.

7 StackGAN

StackGAN — порождающая состязательная сеть для генерации фото-реалистичных изображений (256x256) исходя из текстового описания. Генерировать фото-реалистичные изображения на обычных GAN сложно, поэтому была придумана двух-этапная модель генерации. Stage-I GAN рисует скетчи с примитивными формами и цветами, основанные на текстовом описании, в низком разрешении. Stage-II GAN принимает на вход изображения с первого этапа и текстовое описание и генерирует изображение в высоком разрешении с фото-реалистичными деталями. Чтобы улучшить разнообразие синтезированных изображений и стабилизировать обучение, вместо CGAN использовался метод Conditioning Augmentation.

Раннее использовались CGAN, поскольку на вход им можно было подавать условия, но просто добавляя слои, увеличивающие размер изображения, достичь хороших результатов не удалось. Поэтому основной задачей было повысить разрешение изображений.

7.1 Stage I

Одной из ключевых особенностей StackGAN является Conditioning Augmentation, так как оно позволило расширить количество примеров тренировочного сета, путем небольших случайных изменений в исходных изображениях, что увеличивало многообразие данных. Как показано на картинке, текстовое описание t кодировщиком переводится в векторное представление φ_t (рис. 12). Раннее векторное представление нелинейно трансформировалось, чтобы получить скрытые условные переменные, которые подавались на вход генератору, однако простарнство значений скрытых переменных имеет большую размерность, что приводило к разрывам в многообразии данных, что не выгодно для генератора.

Чтобы избавиться от этого как раз нужно Conditioning Augmentation, которое в отличии от предоставления фиксированных значений переменных выбирает их из нормального распределения $\mathcal{N}(\mu(\varphi_t), \Sigma(\varphi_t))$, где t — текстовое описание, а φ_t — векторное представление . В добавок к уже упомянотому, чтобы сделать многообразие гладким и не переобучиться, нужно добавить регуляризацию,$r = D_{KL}(\mathcal{N}(\mu(\varphi_t), \Sigma(\varphi_t)) \,||\mathcal{N}(0, I))$ (Расстояние Кульбака — Лейблера).

Тренировка дискриминатора D_0 и генератора G_0:

$L_{D_0} = \mathbb{E}_{(I_0,t)\sim p_{data}}[\log D_0(I_0, \varphi_t)] + E_{z\sim t, t\sim p_{data}}[\log(1 - D_0(G_0(z, \hat{c_0}), \varphi_t))]$

$L_{G_0} = \mathbb{E}_{z\sim t, t\sim p_{data}}[\log(1 - D_0(G_0(z, \hat{c_0}), \varphi_t))] + \lambda r,$

где реальное изображение I_0 и описание текста t берутся из реального распределения данных p_{data}, z — шумовой вектор.

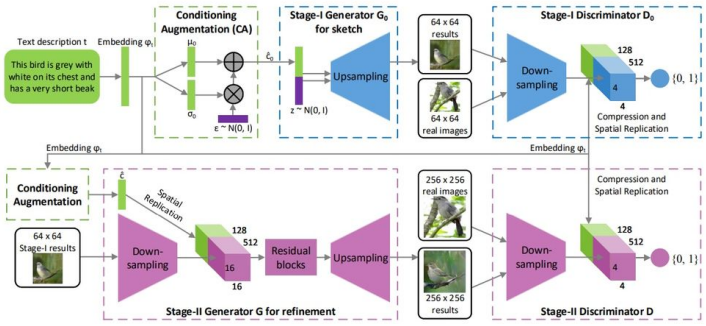

Рис. 12: CGAN

7.2 Stage II

В изображениях с низким разрешеним, сгенерированные Stage-I GAN, обычно недостает ярких деталей и могут быть искривления форм, некоторые детали изображения также могут быть опущены на первом этапе. Stage-II GAN построен над Stage-I GAN и принимает на вход его выход, и текстовое

описание, чтобы исправить и дополнить изображение. Его дискриминатор и генератор тренируются путем поочередной макисимизации L_D и минимизации L_G, как показано в уравнениях:

$$L_D = \mathbb{E}_{(I,t)\sim p_{data}}[\log D(I, \varphi_t)] + \mathbb{E}_{s_0\sim p_{C_0}, t\sim p_{data}}[\log(1 - D(G(s_0, \hat{c}), \varphi_t))]$$

$$L_G = \mathbb{E}_{s_0\sim p_{C_0}, t\sim p_{data}}\left[\log\left(1 - D\left(G(s_0, \hat{c}), \varphi_t\right)\right)\right] + \lambda r,$$

где $s_0 = G_0\left(z, \hat{c_0}\right)$ — результат работы генератора Stage-I GAN.

8 LAPGAN

LAPGAN — генеративная параметрическая модель, представленная пирамидой лапласианов с каскадом сверточных нейронных сетей внутри, которая генерирует изображения постепенно от исходного изображения с низким разрешением к изображению с высоким. На каждом уровне пирамиды обучается сверточная генеративная модель, используя подход порождающих состязательных сетей. Такая стратегия позволяет декомпозировать задачу генерации изображений на последовательность уровней, что упрощает ее решение.

- Пусть $d(\cdot)$ — операция сжатия изображения размера $j \times j$ так, что новое изображение $d(I)$ имеет размеры $j/2 \times j/2$
- $u(\cdot)$ — операция расширения такая, что $u(I)$ имеет размеры $2j \times 2j$.

Тогда пирамида гауссианов имеет вид $\mathcal{G}(I) = [I_0, I_1, \ldots, I_k]$, где $I_0 = I$, и I_k представляет собой k раз выполненное применение $d(\cdot)$.

Коэффициенты h_k на каждом уровне пирамиды:

$$h_k = \mathcal{L}_k(I) = \mathcal{G}_k(I) - u(\mathcal{G}_{k+1}(I)) = I_k - u\left(I_{k+1}\right)$$

9 Сегментация

Вновь обратимся к задаче сегментации. Пусть $X \in \mathbf{R}^{m\times n\times d}$ — изображение, $f : X \to Y$, $Y \in [0, 1]^{m\times n}$, $Y_{ij} = \mathsf{P}(X_{ij} \in \text{segment})$ — функция, сопоставляющая изображению его маску, segment — множество принадлежащих сегменту пикселей. Решают задачу *supervised*-методами, то есть требуется наличие размеченных изображений. Чаще всего для решения этой задачи используется архитектура сети типа *encoder-decoder*. Как следует из названия, сеть состоит из двух частей — *encoder*, который последовательно уменьшает размер изображения (кодирует), и *decoder*, который восстанавливает исходное изображение из закодированного (декодирует). Обычно слои сети симметричны относительно момента перехода от кодировщика к декодировщику

и состоят из нескольких наборов сверточных слоёв с батч-нормализацией и функцией активации, но для кодировщика после каждого такого набора также стоит пулинг-слой, а для декодировщика — наоборот, все это предваряется *upsampling*-слоем. На рисунке 13 представлена схема устройства сети SegNet, предназначенной для решения задачи сегментации изображений.

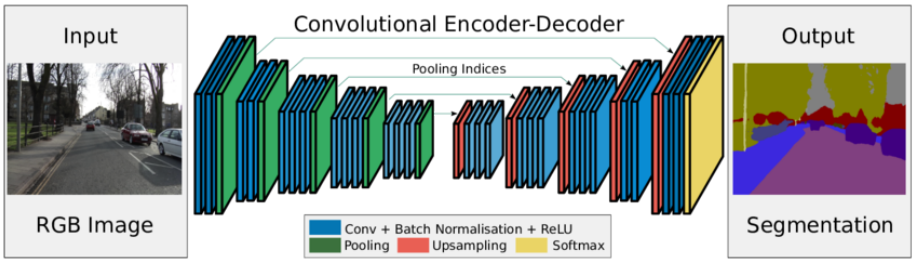

Рис. 13: Схема архитектуры сети SegNet

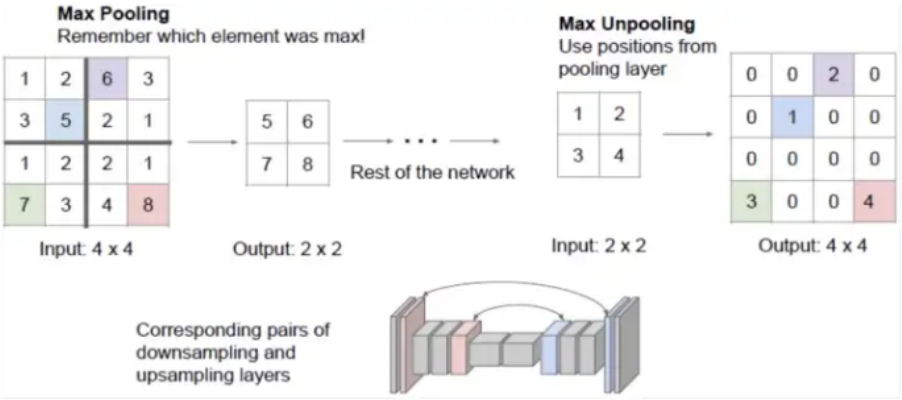

Рис. 14: Иллюстрация работы макс-пулинга и макс-анпулинга.

Обсудим теперь, что же такое *upsampling*-слой. Первый тип *upsampling* называется *unpooling*. Анпулинг пытается выполнить ровно противоположную операцию к пулингу. Можно сделать это разными способами, но чаще всего применяют $max-unpooling$. Во время выполения макс-пулинга сохраняем позиции, где достигается максимум, и затем используем их в unpooling слое, т.е. помещаем элементы на соотвествующие им позиции, все остальное заполняем нулями (Рис. 14).

Одной из возможных модификаций ранее описанной сети является сеть *Vanilla Unet*. Особенностью этой сети является использование данных со слоев кодировщика в соответствующих слоях декодировщика (см. Рис. 15) .

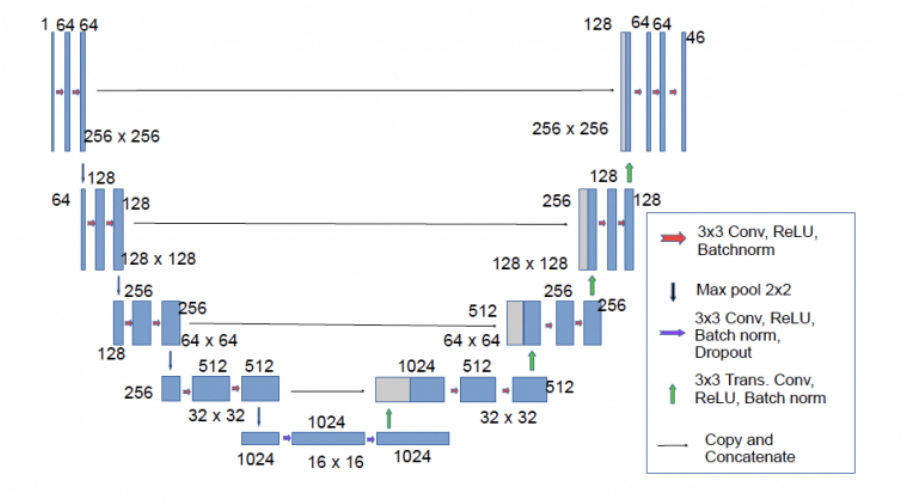

Рис. 15: Схема архитектуры сети Vanilla Unet

В некоторых случаях эта модификация дает значимое улучшение качества сегментации.