ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
имени М.В.ЛОМОНОСОВА»

ФИЗИЧЕСКИЙ ФАКУЛЬТЕТ

КАФЕДРА МАТЕМАТИЧЕСКОГО МОДЕЛИРОВАНИЯ И ИНФОРМАТИКИ

БАКАЛАВРСКАЯ РАБОТА

НЕЙРОСЕТЕВОЙ СИНТЕЗ ТЕКСТУР С ТРЕНДАМИ

Выполнил студент
435 группы:
Будакян Я. С.

_____

Научный руководитель:
к.т.н., доц. Грачев Е. А.

_____

Допущена к защите
Зав. кафедрой _____

Москва

2017

Содержание

ВВЕДЕНИЕ

Здесь, по идее, должно быть общее описание задачи, актуальность и все такое.

1 Постановка задачи

Математически сформулировать поставленную задачу можно с помощью так называемой вероятностной постановки задачи обучения [1, 2]. Рассмотрим многомерное пространство X, содержащее множество всех изображений x: $X = \{x\}$. Тогда обучающая выборка изображений с трендами $D = \{x_i\}$ задает в этом пространстве вероятностное распределение $P_X : X \longrightarrow [0, 1]$, устроенное таким образом, что точки, соответствующие изображениям из выборки, имеют высокую вероятность, а остальные - низкую. Тогда с математической точки зрения задача синтеза текстуры с трендом сводится к синтезу случайного изображения x', принадлежащего распределению, близкому к задаваемому обучающей выборкой:

$$P_{X'} \approx P_X, \quad x' \sim X'$$

"Классический"статистический подход к решению подобного рода задач заключается в введении в рассмотрение параметризированного семейства распределений вероятности и его подстройке на имеющихся данных:

- Вводится параметризированное семейство распределений вероятности $P_\theta(x)$
- Параметры θ находятся из обучающей выборки:

$$\mathcal{L}_\theta(D) = \prod_{x \in D} P_\theta(x)$$

$$\theta^* = \arg\max_\theta \mathcal{L}_\theta(D)$$

- Генерируется объект(изображение) из распределения P_{θ^*}

Этот подход приводит к проблемам:

- Пространство параметров θ может быть огромной размерности

- Известной параметрической модели распределения может вообще не существовать

Простой пример объекта со сложным пространством параметров - человеческое лицо. Задачу генерации изображения реалистичного человеческого лица долгое время не могли решить с удовлетворительным качеством. Однако последние достижения в области искуственных нейронных сетей привели к существенному повышению качества генеративных моделей самого разнообразного типа. Собственно, наличие впечатляющих работ последних лет в этой области ***тут цитаты*** и мотивирует попытаться применить современные нейросетевые подходы в поставленной задаче.

2 Нейронные сети

Искуственная нейронная сеть - это математическая модель, построенная по принципу организации и функционирования биологических нейронных сетей. Она представляет из себя систему соединенных простых блоков - искуственных нейронов, каждый из которых имеет входы и выходы для взаимодействия с другими нейронами. Главное преимущество нейронных сетей перед традиционными алгоритмами в том, что они обучаются на некотором наборе данных, а не программируются в классическом смысле этого понятия. Процесс обучения заключается в нахождении оптимальных весовых коэффициентов между нейронами. С математической точки зрения, процесс обучения - это задача многопараметрической нелинейной оптимизации.

2.1 Математическая модель нейрона

Одиночный нейрон обычно представляет собой взвешенный сумматор с нелинейной функцией активации на выходе (Рис. 1):

$$x_{out} = \phi(\vec{w} \cdot \vec{x_{in}}),$$

где $\vec{w}$ - вектор весовых коэффициентов связей, $\vec{x_{in}}$ - входной вектор, ϕ - нелинейная функция активации.

Функция активации может выбираться разной в зависимости от задачи. Наиболее часто используемые функции:

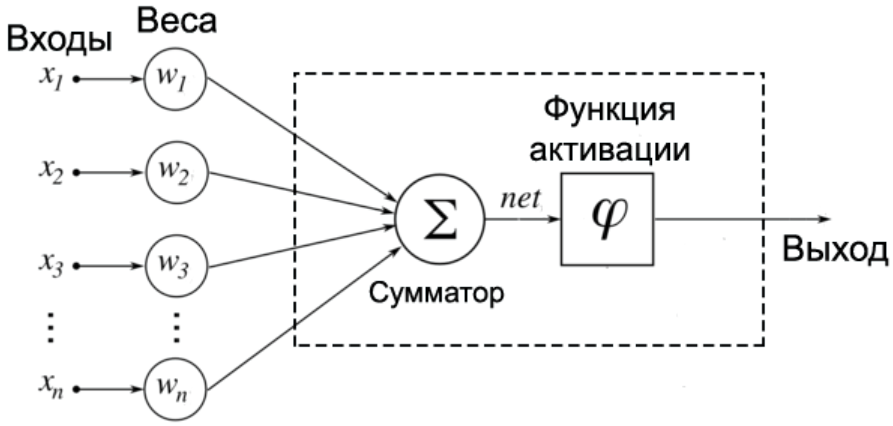

Рис. 1: Математическая модель нейрона

- Сигмоида (логистическая функция)

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

- Гиперболический тангенс

- ReLU

$$ReLU(x) = \max(0, x)$$

- softmax

$$\sigma(\vec{x})_j = \frac{e^{x_j}}{\sum_{k=1}^{N} e^{z_k}}$$

Множество таких нейронов объединяется в сеть и обучается каким-либо методом оптимизации.

2.2 Метод обратного распространения ошибки

Метод обратного распространения ошибки (backpropagation) - самый широко используемый и успешный алгоритм обучения глубоких (многослойных) нейронных сетей. Суть этого метода заключается в распространении сигналов ошибки от выходов сети к ее входам в обратном к распространению сигнала в сети направлении. Это позволяет вычислить производные ошибки по весам сети, которые потом можно использовать в любом градиентном алгоритме оптимизации (например, в стохастическом градиентном спуске).

Обозначим множество входов сети как $\{x_1, \ldots, x_n\}$, множество выходов - O, w_{ij} - вес, присвоенный ребру, соединяющему i-й и j-й узлы, y_k - известные (правильные) ответы, o_i - выход i-го узла. Введем функцию ошибки (например, сумма квадратов расстояний):

$$L(\vec{x}, W) = \frac{1}{2}\sum_{k \in O}(y_k - o_k)^2,$$

где $W = \{w_{ij}\}$ - матрица весовых коэффициентов

Рассмотрим сначала нейроны последнего слоя. Весовой коэффициент w_{ij} влияет на выход сети как часть суммы $S_j = \sum_i w_{ij}x_i$. Соответственно,

$$\frac{\partial L}{\partial w_{ij}} = \frac{\partial L}{\partial S_j}\frac{\partial S_j}{\partial w_{ij}} = x_i\frac{\partial L}{\partial S_j}$$

Аналогично, S_j влияет на общую ошибку только в рамках выхода j-го узла o_j, поэтому

$$\frac{\partial L}{\partial S_j} = \frac{\partial L}{\partial o_j}\frac{\partial o_j}{\partial S_j} = (\frac{\partial}{\partial o_j}\frac{1}{2}\sum_{k \in Out}(y_k - o_k)^2)(\frac{\partial \phi(S)}{\partial S}\Big|_{S=S_j})$$

Если узел j не находится на последнем слое, то у него есть набор связей с нейронами следующего слоя. Обозначим их множество как K_j. Тогда

$$\frac{\partial L}{\partial S_j} = \sum_{k \in K_j}\frac{\partial L}{\partial S_k}\frac{\partial S_k}{\partial S_j}$$

$$\frac{\partial S_k}{\partial S_j} = \frac{\partial S_k}{\partial o_j}\frac{\partial o_j}{\partial S_j} = w_{jk}\frac{\partial o_j}{\partial S_j}$$

$\frac{\partial L}{\partial S_k}$ - аналогичная поправка, но для нейрона следующего слоя. В итоге, получены выражения для производных ошибки по весам для нейронов выходного слоя, а аналогичные производные для нейронов внутренних слоев выражены через нейроны следующих слоев. Это и есть процесс обратного распространения ошибки - градиенты ошибки по весам вычисляются последовательно, начиная с выходного слоя и заканчивая первым.

3 Сверточные нейронные сети

3.1 Сверточная арифметика

3.2 Сверточные слои

4 Генеративные состязательные сети

Архитектура нейронной сети, получившая название генеративной состязательной сети (generative adversarial network - GAN), впервые была описана в 2014 году [2]. В последние 2 года сети такого типа добились больших успехов в задачах синтеза объектов из сложных распределений (например, лиц) [4], переноса стиля [3] и подобных. Этим объясняется мотивация попытки применения данной архитектуры для решения поставленной задачи.

4.1 Общая структура

Переформулируем изначальную задачу нахождения такой процеруды генерирования X', чтобы $P_{X'} \approx P_X$:

$$\rho(P_{X'}, P_X) \longrightarrow \min_{P_{X'}}$$

Введем параметризированную процедуру генерации:

$$X' = g_\theta(\cdot)$$

Переформулируем:

$$\rho(P_{X'}, P_X) \longrightarrow \min_{P_{X'}}$$

$$\rho(g_\theta(\cdot), P_X) \longrightarrow \min_{g_\theta(\cdot)}$$

$$\rho(g_\theta(V), P_X) \longrightarrow \min_{\theta}$$

Возникает вопрос: что использовать в качестве метрики похожести двух распределений ρ, где одно из распределений задано обучающей выборкой. В качестве такой метрики можно использовать функцию потерь обученного классификатора, потому что естественно предположить, что чем чаще ошибается

обученный классификатор, тем больше одно распределение похоже на другое. Тогда задача примет вид:

$$\rho(P_{X'}, P_X) \longrightarrow \min \Leftrightarrow L \longrightarrow \max,$$

где L - функция потерь обученного классификатора. Соответственно, можно ввести две нейросети:

- $d_\zeta(x)$ - классификатор для измерения расстояния, **'дискриминатор'**
- $g_\theta(x)$ - сеть, трансформирующая шум в X', **'генератор'**

Суть использования двух сетей состоит в том, что они обучаются совместно, конкурируя друг с другом: генератор пытается имитировать целевое распределение, а дискриминатор пытается классифицировать поступающие от генератора и из обучающей выборки изображения на 2 класса: реальные (из изначального распределения P_X) и ложные (из $P_{X'}$, т.е. произведенные генератором). Для дальнейшего рассмотрения введем функцию потерь дискриминатора(например, logloss):

$$l_1 = l(d_\zeta(x), 1) \text{ - ошибка 1 рода}$$

$$l_2 = l(d_\zeta(x'), 0) \text{ - ошибка 2 рода}$$

$$L(X, X') = \frac{1}{2}\mathbb{E}_X l_1 + \frac{1}{2}\mathbb{E}_{X'} l_2 = -\frac{1}{2}(\mathbb{E}_X \log d_\zeta(x) + \mathbb{E}_{X'} \log(1 - d_\zeta(x'))) =$$

$$= -\frac{1}{2}(\mathbb{E}_X \log d_\zeta(x) + \mathbb{E}_V \log(1 - d_\zeta(g_\theta(v)))) = L(\zeta, \theta).$$

Функция потерь обученного классификатора:

$$L^*(\theta) = \min_\zeta L(\zeta, \theta)$$

Соответственно,

$$\min_\zeta L(\zeta, \theta) \longrightarrow \max_\theta$$

$$\theta^* = \arg\max_\theta \left[\min_\zeta L(\zeta, \theta)\right]$$

Определим оптимальный дискриминатор:

$$d_\theta^* = d_{\zeta^*(\theta)}$$

$$\zeta^*(\theta) = \arg\min_\zeta L(\zeta, \theta)$$

4.2 Обучение GAN

Итак, задача обучения GAN свелась к нахождению

$$\theta^* = \arg\max_\theta \left[\min_\zeta L(\zeta, \theta)\right]$$

Решить ее можно, например, методом стохастического градиентного спуска:

$$\Delta\theta \sim \nabla L(\zeta^*(\theta), \theta)$$

Для малых изменений $\Delta\theta$:

$$\nabla L(\zeta^*(\theta), \theta) \approx \nabla L(\zeta^*(\theta), \theta + \Delta\theta)$$

В итоге, процесс обучения принимает следующий вид:

- Обучаем дискриминатор при фиксированном генераторе
- Обучаем генератор при фиксированном дискриминаторе
- Повторяем до сходимости параметров обеих моделей

4.3 Различные модификации

4.3.1 pix2pix GAN

Для решения задачи было попробовано применить модификацию GAN-сети под названием "pix2pix GAN"[6]. Ее отличие от схемы GAN, введенной выше, состоит в том, что вместо шума на вход генератору приходят другие изображения, на которых он основывается при синтезе. Схематически ее устройство изображено на (Рис. 3). Для pix2pix сети общий функционал

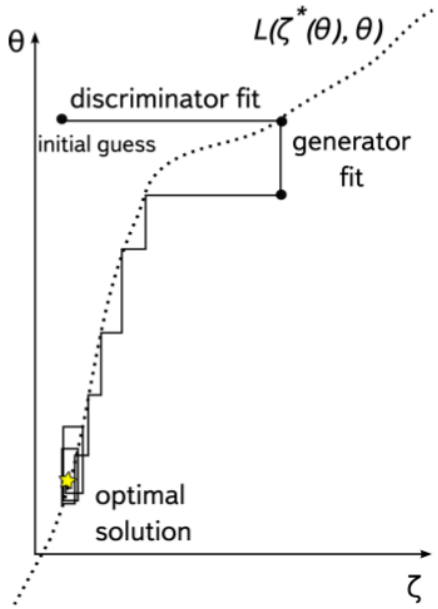

Рис. 2: Схематическое изображение процесса обучения GAN.

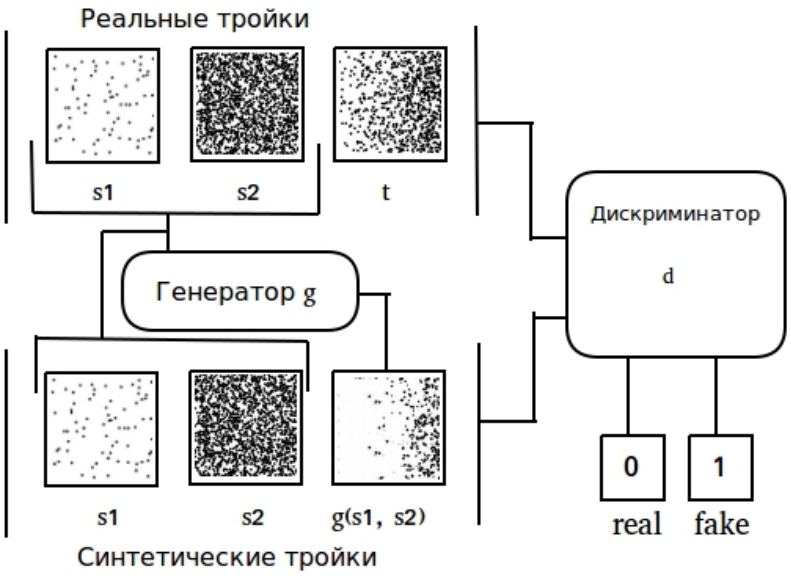

Рис. 3: Схематическое устройство сети pix2pix GAN.

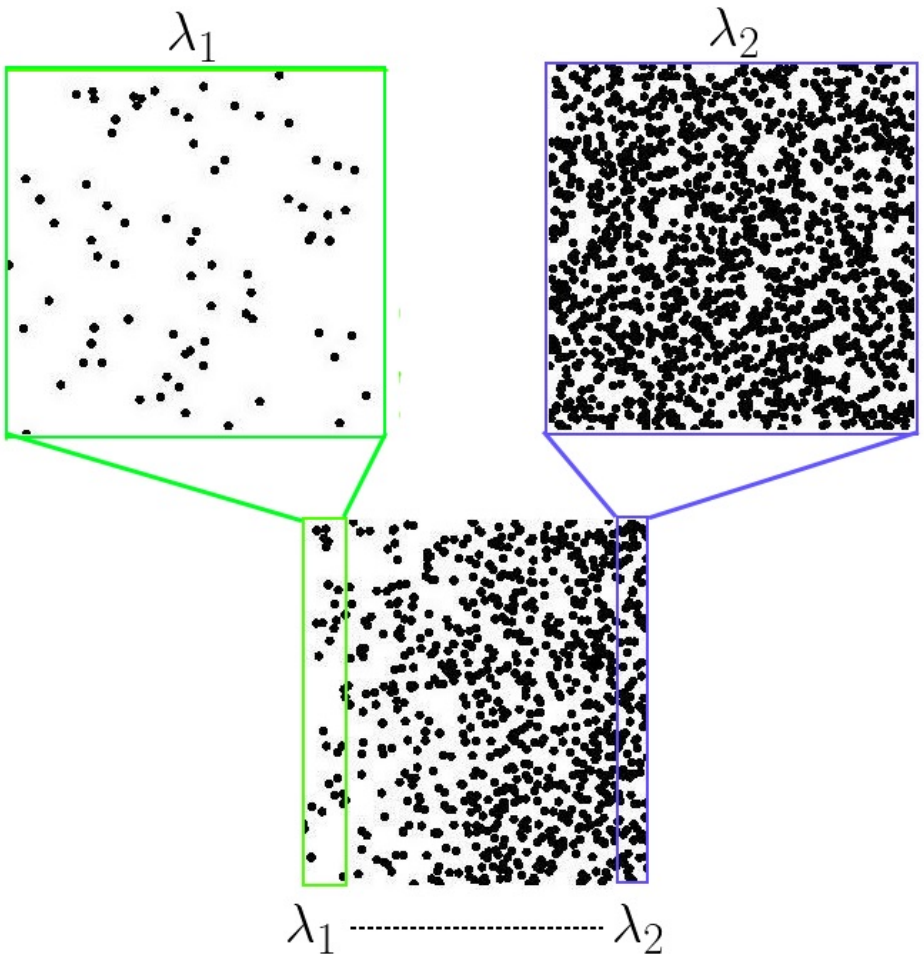

Рис. 4: Вход и желаемый выход нейросети-генератора.

потерь выглядит следующим образом:

$$L(G, D) = L_{adv}(G, D) + \eta \mathbb{E}_{p_{data}(s_1,s_2,r)}(\| r - G(s_1, s_2) \|_1)$$

$$L_{adv}(G, D) = \mathbb{E}_{p_{data}(s_1,s_2,r)} \log D(s_1, s_2, r) + \mathbb{E}_{p_{data}(s_1,s_2)} \log(1 - D(s_1, s_2, G(s_1, s_2)))$$

где G, D - генератор и дискриминатор, (s_1, s_2, r) - тройка изображений (интенсивность слева, справа и реальное изображение с трендом), $\mathbb{E}_{p_{data}(s_1,s_2,r)}$ - мат. ожидание логарифмического правдоподобия того, что тройка изображений (s_1, s_2, r) принадлежит вероятностному распределению реальных троек $p_{data}(s_1, s_2, r)$, а $p_{data}(s_1, s_2)$ соответствует распределению реальных изображений s_1, s_2.

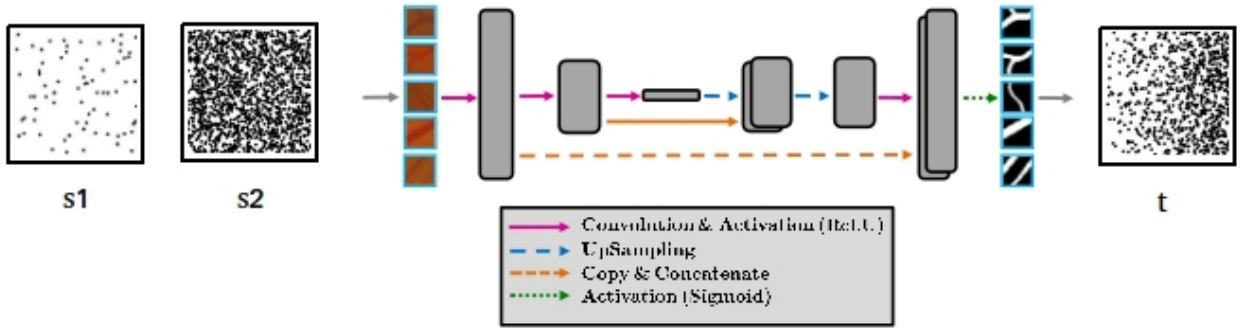

Рис. 5: Схематическое изображение нейросети-генератора.

5 Синтез текстур

6 Методы стохастической оптимизации

6.1 Adam

6.2 RMSprop

7 Оценка качества синтеза

После обучения сети, необходимо проверить, что сгенерированные ей изображения действительно имеют искомые характеристики. Для этого нужно ввести специальную метрику, которая будет учитывать наличие в изображении тренда интенсивности частиц. Было решено использовать среднюю плотность черных пикселей в некотором окне, и проходить этим окном по изображению (Рис. 6):

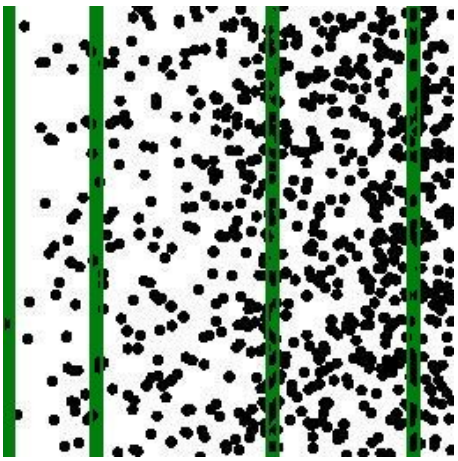

Рис. 6: Прохождение окном, W, H - размеры изображения, w - ширина окна.

$$\xi_k = \frac{1}{Hw}\sum_{i=k}^{k+w}\sum_{j=0}^{H}\left|\frac{x(i,j)-255}{255}\right|,$$

$$k = \overline{1, W-w+1}$$

Построив график $\xi(k)$, можно увидеть, как меняется плотность пикселей и прослеживается ли тренд. В качестве метрики можно взять среднеквадратичную ошибку:

$$\xi = \frac{1}{W-w}\sum_{k=1}^{W-w+1}(\xi_k - \xi_{0k})^2,$$

где ξ_{0k} - это ξ_k, усредненное по примерам из обучающей выборки. Соответственно, чем меньше значение метрики, тем лучше тренд, присутствующий на сгенерированном изображении, приближает искомый.

8 Результаты

Описанные подходы были реализованы в виде компьютерных программ на языке Python с помощью библиотеки для построения искусственных нейронных сетей Keras [13], который, в свою очередь, привлекает для расчетов библиотеку Tensorflow [14]. Обучение проводилось на синтетических данных.

8.1 Выборка с одним трендом

Выборка состояла из 6000 обучающих изображений и 316 валидационных. Все изображения содержали в себе различные случайные реализации одного тренда интенсивности

$$\lambda(x) =$$

тут картинки примеров из обучающей выборки

8.1.1 GAN

8.1.2 Синтез текстур

8.2 Выборка с множеством трендов

8.2.1 GAN

8.2.2 Синтез текстур

ЗАКЛЮЧЕНИЕ

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] Воронцов К. В., "Математические методы обучения по прецедентам (теория обучения машин)".

[2] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bign Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, Yoshua Bengio, "Generative Adversarial Nets"// arXiv: 1406.2661 [stat.ML], 2014.

[3] Leon A. Gatys, Alexander S. Ecker, Matthias Bethge, "A Neural Algorithm of Artistic Style"// arXiv: 1508.06576 [cs.CV], 2015.

[4] Jon Gauthier, "Conditional generative adversarial nets for convolutional face generation Tech. rep., 2015.

[5] Mehdi Mirza, Simon Osindero, "Conditional Generative Adversarial Nets"// arXiv: 1411.1784 [cs.LG], 2014.

[6] Pedro Costa, Adrian Galdran, Maria Inês Meyer, Michael David Abràmoff, Meindert Niemeijer, Ana Maria Mendonça, Aurélio Campilho, "Towards Adversarial Retinal Image Synthesis"// arXiv: 1701.08974 [cs.CV], 2017.

[7] Aäron van den Oord, Nal Kalchbrenner, Oriol Vinyals, Lasse Espeholt, Alex Graves, Koray Kavukcuoglu, "Conditional Image Generation with PixelCNN Decoders"// arXiv: 1606.05328 [cs.CV], 2016.

[8] Augustus Odena, Christopher Olah, Jonathon Shlens, "Conditional Image Synthesis with Auxiliary Classifier GANs"// arXiv: 1610.09585 [stat.ML], 2016.

[9] Tejas D. Kulkarni, Will Whitney, Pushmeet Kohli, Joshua B. Tenenbaum, "Deep Convolutional Inverse Graphics Network"// arXiv: 1503.03167 [cs.CV], 2015.

[10] Junbo Zhao, Michael Mathien, Yann LeCun, "Energy-based Generative Adversarial Networks"// arXiv: 1609.03126 [cs.LG], 2016.

[11] Chuan Li, Michael Wand, "Combining Markov Random Fields and Convolutional Neural Networks for Image Synthesis"// arXiv: 1601.04589 [cs.CV], 2016.

[12] Dmitry Ulyanov, Vadim Lebedev, Andrea Vedaldi, Victor Lempitsky, "Texture Networks: Feed-forward Synthesis of Textures and Stylized Images"// arXiv: 1603.03417 [cs.CV], 2016.

[13] François Chollet, Keras, 2015. Software available from github.com/fchollet/keras.

[14] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Rafal Jozefowicz, Yangqing Jia, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Mike Schuster, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke,

Yuan Yu, and Xiaoqiang Zheng, TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

ПРИЛОЖЕНИЕ

Использованные версии программных пакетов

- Python 3.6
- Tensorflow 1.1.0
- Keras 2.0.4