Правительство Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования «Санкт-Петербургский государственный
университет»
Математическое обеспечение и администрирование информационных
систем

Кафедра информационно-аналитических систем

Полянский Владислав Владимирович

Использование видео-последовательностей для предобучения сверточных искусственных нейронных сетей

Бакалаврская работа

Научный руководитель:
доцент, к.ф.-м.н. Михайлова Е. Г.

Рецензент:
Мордвинцев А. С.

Санкт-Петербург
2016

SAINT-PETERSBURG STATE UNIVERSITY

Sub-Department of Analytical Information Systems

Polianskii Vladislav

Usage of video-sequences for convolutional neural network pre-training

Graduation Thesis

Scientific supervisor:
associate professor, PhD Elena Mikhailova

Reviewer:
Alexander Mordvintsev

Saint-Petersburg
2016

Оглавление

Введение

В настоящее время очень активно развиваются и имеют большое применение в анализе видео и изображений алгоритмы, построенные на сверточные искусственных нейронных сетях[7]. Чаще всего они применяются при классификации изображений и распознавании образов. Большинство подобных сетей используют обучение с учителем. В то же время обучение без учителя получает значительно меньше внимания ввиду определенных сложностей реализации и недостаточной изученности данной области. Тем не менее, за последние несколько лет стали активно появляться новые методы, так или иначе задействующие обучение без учителя. В данной работе проводится исследование на тему обучения нейронной сети, выполняющей задачу определения динамики на входном изображении, а также задачу предсказания (генерации) следующего состояния запечатленного участка с учетом изменяющихся динамических объектов на изображении.

Генерация изображений может быть применена к моделированию и анализу всевозможных естественных структур, таких как медицинские данные, данные спутниковой съемки, картографические данные. Возможно применение подобных идей в сфере безопасности, подобная модель могла бы помочь в обнаружении движения по одному снимку с камеры еще до того, как оно само было совершено. Также, задачу генерации следующего изображения можно рассматривать с творческой стороны; при последовательном применении операции генерирования следующего кадра можно придать динамичность статичной фотографии, превратив ее в анимацию.

1. Постановка задачи

Цель данной работы - исследовать возможности сверточных нейронных сетей для определения движения на последовательных кадрах видео и генерации следующего, исходя только из одного последнего снимка. В частности, в цели работы входят использование последних наработок в областях анализа работы обученной сети и генерации изображений.

Проводимая в данной статье работа разбивается на два основных этапа:

1. Построение предикативной модели нейронной сети. На вход поступают два изображения, предположительно взятых из некоторого видеопотока. Пусть на первом изображении запечатлен объект (например, городская улица) в некоторый момент времени t. Алгоритм должен определить, верно ли, что второе изображение описывает тот же объект в момент времени $t + \Delta t$, где Δt положительно и сравнительно мало. Другими словами, если изображения являются двумя кадрами одного видео, то алгоритм должен понять, верно ли, что второй кадр следует вскоре после первого.
   Предполагается, что во всех видеопотоках камера расположена статично, а на записи присутствуют динамичные объекты (например, проезжающие автомобили).

2. После построения и обучения вышеописанного алгоритма возникает идея развернуть его в генеративном направлении. По *одному* изображению можно попытаться сгенерировать новое - следующий кадр предполагаемого видеоряда, то есть наиболее вероятный снимок, который мог бы произойти через совсем небольшое время. Эта часть, ввиду генерации изображения, комбинирует в себе как обучение с учителем, так и обучение без учителя.

В каждом этапе помимо построения и обучения самих моделей также выполняется анализ работы сетей при помощи визуализации работы внутренних нейронов.

2. Описание предметной области

В последнее время всё в большем объеме при построении моделей, связанных с классификацией, анализом, преобразованием данных, представляемых в виде изображений, используются модели сверточных нейронных сетей. Так, например, данные сети используются для классификации изображений и видео соответственно в [5] и в [6]. В данном разделе рассмотрено общее устройство сверточных нейронных сетей, а также описание работы состязательной сети, используемой в данной работе.

2.1. Сверточные нейронные сети

Сверточная нейронная сеть являются расширением классических рекуррентных нейронных сетей. Такие сети представляет из себя последовательность слоев, состоящих из нейронов; на первый слой подаются входные данные, последний слой является выходным, а все промежуточные являются полносвязными - каждый нейрон внутреннего слоя соединен взвешенным ребром с каждым нейроном предыдущего слоя. По сути, на внутреннем слое вектор выходов предыдущего слоя умножается на некоторую матрицу весов полносвязного слоя, после чего к полученным значениям применяется некоторая функция активации, позволяющая ввести нелинейность операций.

Сверточные нейронные сети отличаются от классических тем, что в них могут присутствовать не только полносвязные слои. В основные виды слоев также добавляются сверточные и pooling-слои.

2.1.1. Сверточный слой

Сверточный (convolutional) слой является аналогом применения нескольких фильтров к текущему изображению, где под изображением подразумевается выход предыдущего слоя. Для данного слоя определяется его ядро (kernel) размера $M \times N \times C \times F$. F - количество каналов выходного изображения, то есть количество фильтров. $M \times N \times C$ - размер

каждого фильтра, где M и N - соответственно ширина и высота окна фильтра, а C - количество каналов входного изображения. Таким образом, результат самой свертки формируется следующим образом:

$$conv[a, b, f] = \sum_{i=0}^{M} \sum_{j=0}^{N} \sum_{c=0}^{C} input[a - i - {}^{M}/_{2}, b - j - {}^{N}/_{2}, c] \cdot filter[a, b, c, f] \quad (1)$$

В результате, каждый канал выходного слоя является своего рода картой, отображающей наличие определенного признака в областях значений предыдущего слоя.

После свертки обычно также производится прибавление некоторого смещения - константы, одинаковой для всех значений полученной матрицы. Однако при описанной далее нормализации смещение иногда опускается.

Последним этапом свертки является применение функции активации ко всем значениям полученной матрицы. При работе с изображениями в качестве функции активации вместо $tanh$ практически всегда применяется функция $ReLu(x) = max(0, x)$. Основная причина выбора данной функции заключается в том, что при обучении глубоких сетей не возникает проблемы крайне маленького градиента при больших x, в отличие от функции $tanh(x)$, градиент которой стремится к 0 при увеличении x. Также благодаря обрубанию отрицательных значений $ReLu$ увеличивает разреженность значений у внутренних слоев сети, что хорошо обеспечивает нелинейность, а также полезно с вычислительной точки зрения.

В итоге, свертка эквивалентна применению формулы:

$$output = activation(input * filter + bias) \quad (2)$$

, где $input$ - входной слой, $filter$ - ядро свертки, $bias$ - смещение и $output$ - результат.

2.1.2. Pooling

Pooling-слой служит для уменьшения размера матрицы. Данный слой способствует поддержанию инвариантности сети к масштабу, а также позволяет искать более глобальные признаки в изображении.

Основным используемым методом pooling'а является max-pooling. Входной слой разбивается на клетки заданного размера и из каждой клетки берется максимальное значение (иногда берется среднее значение по клетке). Таким образом, при размере окна $M \times N$ изображение уменьшается в $M \cdot N$ раз.

2.1.3. Нормализация

В данной работе применяются два метода нормализации: local response normalization и batch normalization. Обе нормализации используются для предотвращения замедления скорости обучения параметров сети.

Local response normalization является самостоятельным слоем сети. Данная операция нормализует каждое значение входной матрицы по каналу, то есть применяется следующая формула:

$$output[a,b,c] = \frac{input[a,b,c]}{k + \alpha(\sum\limits_{i=-n/2}^{n/2} input[a,b,c+i]^2)^{\beta}} \tag{3}$$

, где n - ширина окна нормализации, k, α и β - другие настраиваемые параметры.

Batch-normalization, впервые введенная в [4], применяет стандартную нормализацию к получаемым значениям, а затем линейно трансформирует:

$$y = \frac{x - \mu}{\sqrt{\sigma^2 + \epsilon}} \cdot \gamma + \beta \tag{4}$$

Здесь ϵ, γ и β являются настраиваемыми параметрами. μ и σ, среднее значение и дисперсия, зависят от того, на каком этапе сейчас находится сеть. Если происходит обучение сети, то эти значения берутся из значений, полученных только в текущий момент. Во время тестирования

сети значения берутся из всей собранной статистики.

Batch-нормализация применяется сразу после применения операции свертки и до применения операции нелинейности (функции активации).

2.2. Генеративные состязательные сети

Понятие генеративных состязательных сетей (Generative Adversarial Networks) было введено в [3]. Этот тип сети используются для генерации изображений, которые сама сеть не может отличить от настоящих. Применение данных сетей широко исследовалось в [8] для генерации различных изображений цифр, тематических изображений и лиц. В работе предлагалось использовать две нейронные сети: генератор и дискриминатор.

Изначально сеть-генератор получает на вход некоторый случайный вектор или матрицу небольшого размера и, постепенно применяя операцию *развертки*, возвращает сгенерированное изображение нужного размера. Операция развертки, которую также начали называть операцией деконволюции после [1], является операцией, применяющей ко входному слою градиент от обычной свертки.

Сеть-дискриминатор принимает на вход изображение и пытается определить, взято ли это изображение из обучающей выборки, или это изображение, полученное из генератора. Таким образом, дискриминатор является обычной сверточной нейронной сетью, классифицирующей изображения на два класса: $real$ и $fake$.

Функции, минимизируемые сетью, для генератора G и дискриминатора D можно записать примерно следующим образом:

$$L_G(z_0) = H(D(G(z_0)), real)$$

$$L_D(\theta, z_0) = H(D(G(z_0)), fake) + H(D(\theta), real)$$

Здесь $H(input, class)$ - функция кросс-энтропии, применяемая как функция потерь при классификации, z_0 - подаваемый случайный вектор, θ - изображение из обучающей выборки (обучение происходит одновременно по настоящим и сгенерированным изображениям).

3. Связанные исследования

В упомянутой ранее работе [8] было проведено исследование на тему методов обучения генеративных сверточных нейронных сетей без учителя. Среди всего, что было сделано в данной работе, были введены определенные ограничения на структуру используемых нейронных сетей. В частности, обычно используемые pooling-слои сверточных нейронных сетей были заменены на аналог сверточных слоев, которые при этом также уменьшают количество вершин в слое. Тем самым появляется возможность также обучать сжимающие слои нейросети (в противоположность стандартому методу max-pooling, который детерминирован и не имеет обучаемых параметров, таких как веса ребер).

В [2] происходит генерация оптического потока по паре изображений. В работе рассматриваются два подхода: в первом изображения сразу объединяются в одно шестиканальное, во втором к изображениям сначала применяются несколько сверточных слоев, потом вычисляется корреляция полученных матриц фич и далее продолжаются обычные сверточные слои. После последнего сверточного слоя начинается генерация изображения оптического потока при помощи развертычных слоев. Помимо развертки, на каждом этапе для повышения точности результата к имеющейся матрице присоединяется матрица фич такого же размера, полученная ранее при свертке. Подобный способ генерации также применяется в настоящей работе на этапе генерации следующего кадра.

В [9] описана процедура автоматического преобразования при помощи сверточных нейросетей обычного двухмерного изображения в трехмерное, на которое можно в дальнейшем посмотреть при помощи 3D-очков. Самым сложным этапом работы является генерация карты глубин изображения. Ввиду трудности получения данных для обучения из фотографий, обучение происходило по кадрам из фильмов, снятых в 3D-формате.

В работе [10] для понимания работы сети приводится метод визуализации внутренних нейронов. Здесь показывается его важность для

понимания работы сети, обнаружении проблем с неравномерной обученности классификатора, анализе признаков, определяемых в классификации. Подобное исследование было также проведено в известной работе Deep Dream, исследовавшей сеть ImageNet.

4. Используемые технологии

Для построения искусственных нейронных сетей использовалась библиотека *Tensorflow* для Python, разрабатываемая компанией Google, позволяющая задать граф вычислений для выполнения различных операций над входными данными. Вершины графа представляют собой математические операции, а ребрами являются многомерные массивы данных (тензоры), передаваемые между этими операциями. Данная библиотека широко используется для применения методов машинного обучения, и предоставляет простое API для проведения вычислений на CPU и GPU. Так как задача обучения нейронных сетей хорошо распараллеливается и прирост скорости обучения на GPU крайне значителен относительно вычислений на CPU, обучение проводилось на графическом процессоре GeForce GTX 960 с объемом памяти 2Gb.

В обучении сети сильно пригодилась утилита *tensorboard*, предоставляемая вместе с библиотекой. она позволяет в режиме реального времени отслеживать состояние интересующих значений тензоров в сети. Помимо графиков целевых функций и точности при обучении, также крайне полезно было следить за графиками изменения гистограмм ядер слоев сети во времени. При неизменяющихся гистограммах можно было понять, что в конкретных слоях обучение не происходит или происходит плохо и перезапустить сеть с другими параметрами и другой начальной инициализацией значений. Также благодаря гистограммам было видно, что веса сети практически не увеличиваются по модулю, из-за чего отпадала необходимость ввода регуляризации весов.

5. Подготовка данных для обучения

Для проведения исследовательских работ были приготовлены два различных набора изображений: искусственно-сгенерированный и из реального мира. Предполагалось, что первые эксперименты будут проводиться на синтетическом наборе, и при успешных результатах будет производиться работа над более сложными данными.

Все наборы данных были разделены на обучающие и тестовые выборки. Хотя из-за достаточно большого объема данных, а также из-за метода обучения (сеть максимизирует функционал не на всех данных сразу, а на небольших подвыборках) результаты работы описанных далее алгоритмов совпадают для тестовых и обучающих данных.

5.1. Arrows

Arrows представляет собой набор сгенерированных *пар* изображений размера 128×128. В каждой паре изображений присутствует задний и передний план. В качестве задних планов были нарезаны изображения, взятые с сайта flickr.com с тегом landscape и nature, причем в каждой паре задний план одинаковый. На передних планах на первых изображениях пар с случайными параметрами расположения, размера, соотношения сторон, цвета и направления были расположены от одной до трех стрелок. На каждом втором изображении были отображены эти же стрелки, смещенные на небольшое случайное расстояние в направлении своего движения. В дополнение, на изображения был наложен легкий гауссов шум.

Количество пар изображений в наборе - 20000.

Также на начальном этапе был сгенерирован и также использовался более простой набор из пар изображений размера 64×64 со стрелками на черном фоне без шумов в количестве пар приблизительно равном 100000. Для дальнейшего упоминания набора будем использовать название *ArrowsSimple*.

Рис. 1: Примеры изображений наборов *Arrows* и *ArrowsSimple*.

5.2. Kingstreet

Для другого набора данных было взято видео с уличной камеры на King Street в Нью-Брансуик, Канада (`https://www.youtube.com/watch?v=pY7zr5imGZc`). Из всего видео были взяты только первые 7 часов (до момента наступления темноты) и выбранная часть была нарезана на кадры (брался каждый 15-й кадр) размера 768×432. Всего получилось приблизительно 64000 изображений.

Изображения не группировались по парам ввиду того, что пары можно было получить естественным путем, взяв i-е и $(i + k)$-е изображения, k подбиралось исходя из желаемого временного расстояния между кадрами.

Рис. 2: Примеры пар изображений набора *Kingstreet*.

6. Классификация парных изображений

На первом этапе была поставлена задача построить классификатор пар изображений, определяющий направление времени между двумя данными изображениями. Другими словами, пусть дана пара $(img1, img2)$, $img1$ отображает состояние в некоторый момент времени t_0, а $img2$ в момент $t_0 + \Delta t$. В зависимости от того, какой знак имеет Δt, классификатор должен определить данную пару в $+1$ или в -1 класс соответственно ($+1$ класс означает, что первое изображение было раньше второго, т.е. $\Delta t > 0$). Было решено сделать именно классификатор, потому что в дальнейшем набор классов при необходимости можно было бы расширить (например, разбив область значений Δt более чем на два промежутка).

6.1. Построение сети

Сеть, решающая поставленную задачу, представляет из себя несколько сверточных слоев, связанных между собой pooling-слоями и слоями нормализации. В отличие от стандартных структур сверточных нейрон-

ных сетей, в построенной сети полностью отсутствуют полносвязные слои. Это сделано с той целью, чтобы в итоге после последнего сверточного слоя получить некоторую карту, значения в которой будут говорить о принадлежности разных областей изображений разным классам. Тогда коэффициентом принадлежности всего изображения определенному классу будет являться просто сумма всех соответствующих коэффициентов для его областей.

Для двух наборов данных сети различаются лишь количеством сверточных слоев - для бóльших изображений использовалось больше слоев.

Так как изображения приходят парами, то сначала изображения каждой пары соединяются друг с другом по третьему каналу, то есть на вход сети подается набор шестиканальных изображений.

6.2. Обучение сети

При обучении сети на вход подается сразу несколько изображений. В качестве целевой функции берется среднее значение из подсчитанных значений функций для каждого изображения из поднабора. Размерами поднаборов изображений были выбраны 64 для *Arrows* и *ArrowsSimple* и 16 для *Kingstreet*.

Изначально обучение происходило на наборе *ArrowsSimple*. На вход подавались случайные 64 пары изображений. Изначально каждая пара изображений принадлежит классу $+1$, поэтому половина пар была просто случайным образом инвертирована для обучения их принадлежности классу -1. На таких данных сеть практически мгновенно обучалась до точности $> 90\%$. Это логично - так как стрелки были на черном фоне, то все ненулевые значения изображения "важны" для нейронной сети и градиент целевой функции сразу изменяет параметры сети в нужном направлении.

Стоит отметить, что точность 100% получить на данном простом наборе невозможно из-за присутствия плохих пар изображений, для которых нельзя однозначно определить класс (например, стрелка, присутствующая на первом изображении, может отсутствовать на втором,

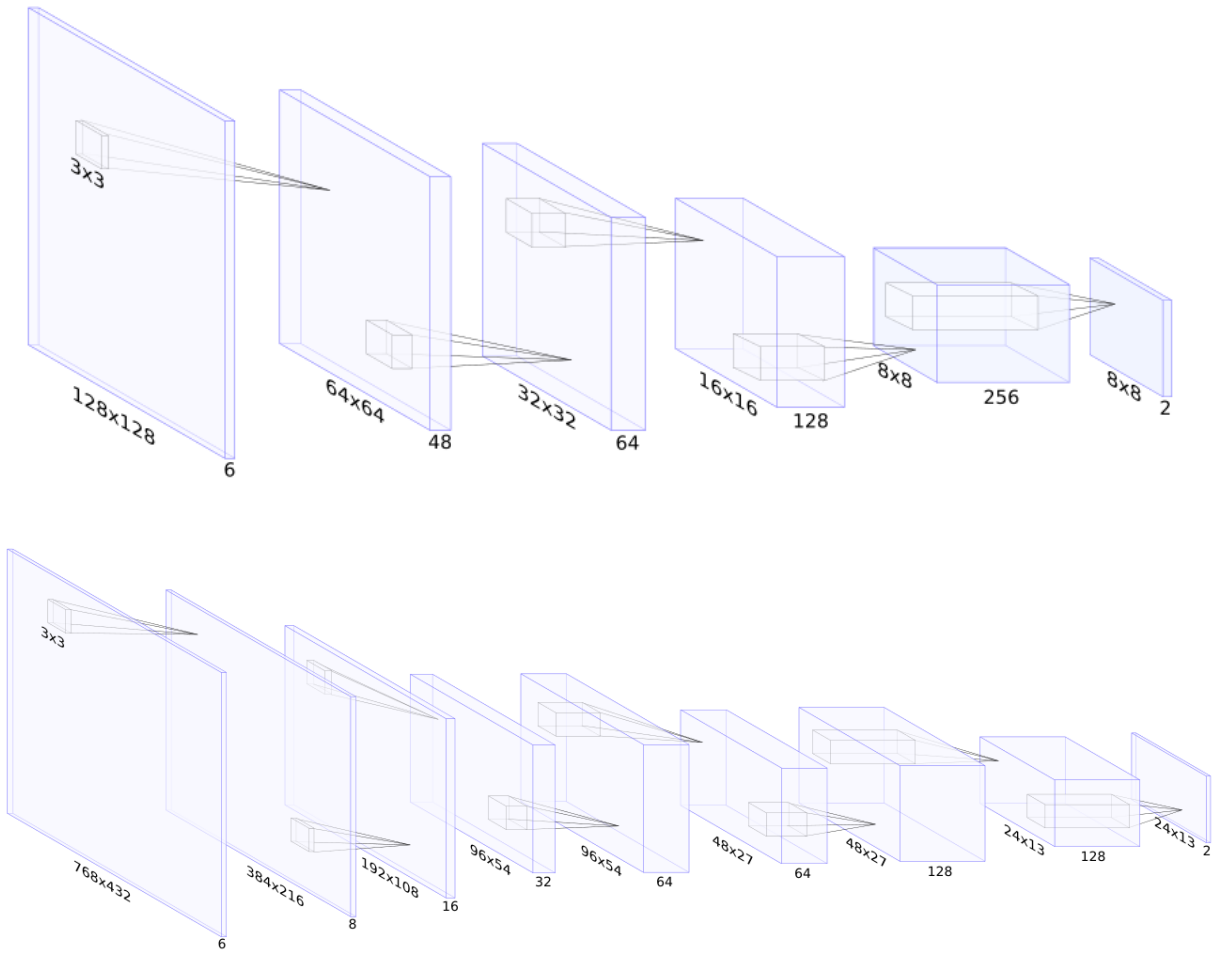


Рис. 3: Структуры классифицирующих сетей *Arrows* и *Kingstreet*. Размер ядра сверточного слоя - 3×3. Размер pooling-окна - 2×2. В местах, где не происходит изменение масштабов изображения, отсутствует pooling.

так как после смещения вперед или назад вышла за границы изображения).

Однако, запуск обучения такой же по структуре сети на наборе *Arrows* не увенчался успехом. К тому же, если применять нормализацию входных изображений (вычет среднего значения пикселей из изображений), то и на *ArrowsSimple* сеть не обучалась за достаточно большое количество итераций.

Заметим, что если $(img1, img2)$ определяется в класс C ($C \in \{-1, +1\}$), то пара $(img2, img1)$ должна определяться в класс $-C$. Тогда есть смысл подавать не N случайных пар изображений, а $\frac{N}{2}$ пар, принадлежащих

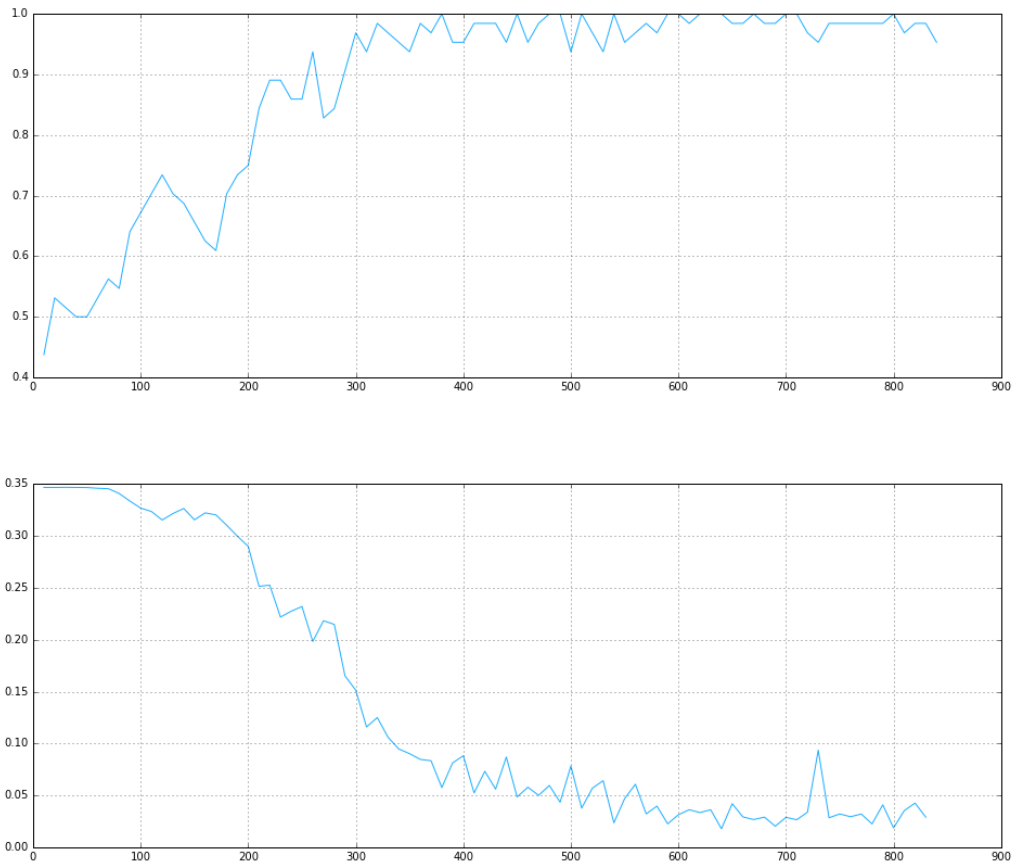


Рис. 4: Графики точности и кросс-энтропии при обучении на наборе *ArrowsSimple*

классу $+1$, и $\frac{N}{2}$ *тех же самых* пар, но с другим порядком изображений внутри пар (то есть принадлежащих классу -1). При подсчете градиентов их значения для пар $(img1, img2)$ и $(img2, img1)$ усредняются, и это навевает на мысль о том, что "неправильные" градиенты будут компенсировать друг друга, а те, что приводят к верной классификации в целом, - только усилятся. Эта идея и правда оказалась верна, и за относительно небольшое количество итераций для *Arrows* и *Kingstreet* были получены точности, приблизительно равные 95%.

6.3. Анализ внутренних слоев сети

После обучения сети возникает идея провести ее анализ. Для этого на данный момент применяются методы визуализации нейронов сети. В качестве параметров сети теперь выступают входные данные, а

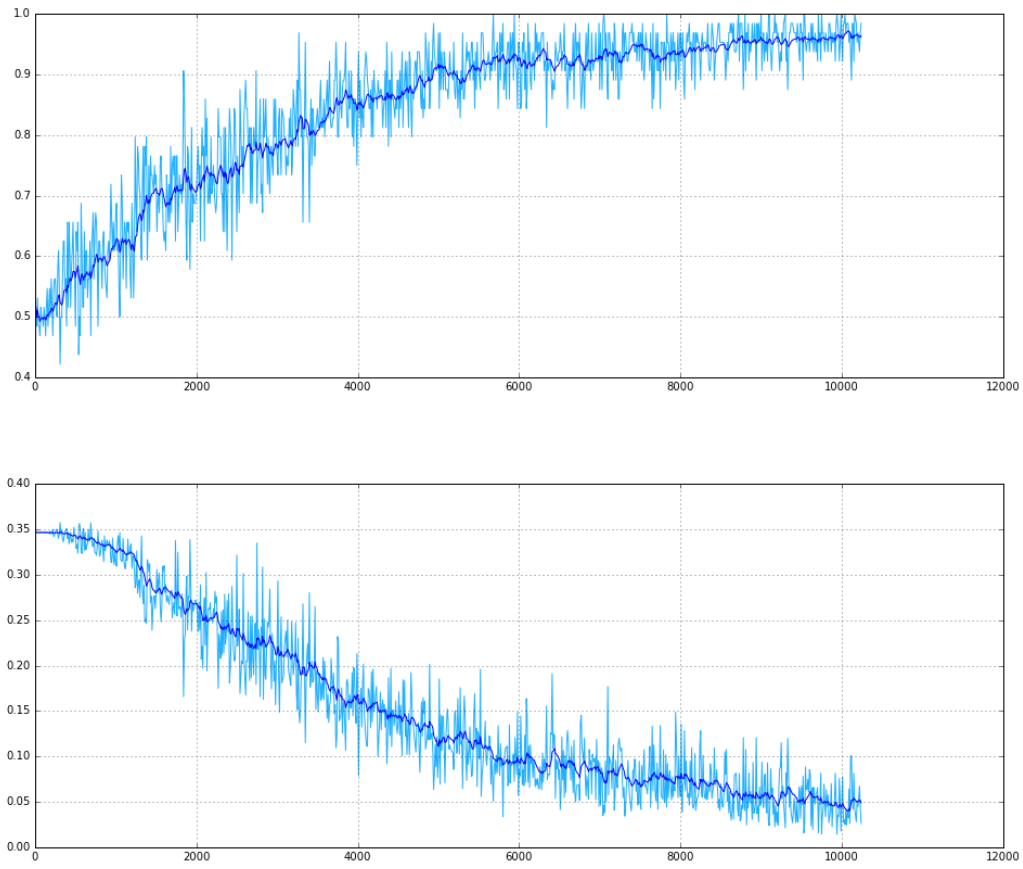


Рис. 5: Графики точности и кросс-энтропии при обучении на наборе *Arrows*

в качестве целевой функции - среднее значение по выводу выбранного нейрона. Таким образом, начиная с некоторого случайного входного изображения (а точнее пары изображений), используя такой же градиентный спуск, как при обучении сети, максимизируется значение выхода нейрона. В результате получается изображение, характеризующее выбранный нейрон.

Известно, что первые слои нейронной сети обычно хорошо распознают прямые линии на изображении (то есть на прямых линиях они выдают большее значение), в то время как на более глубоких слоях появляются более сложные изображения. Также есть некоторое предположение, что у хорошо обученной сети получаются ясные и не случайные изображения нейронов.

Отобразив все нейроны самой первой сети, обученной на *ArrowsSimple*,

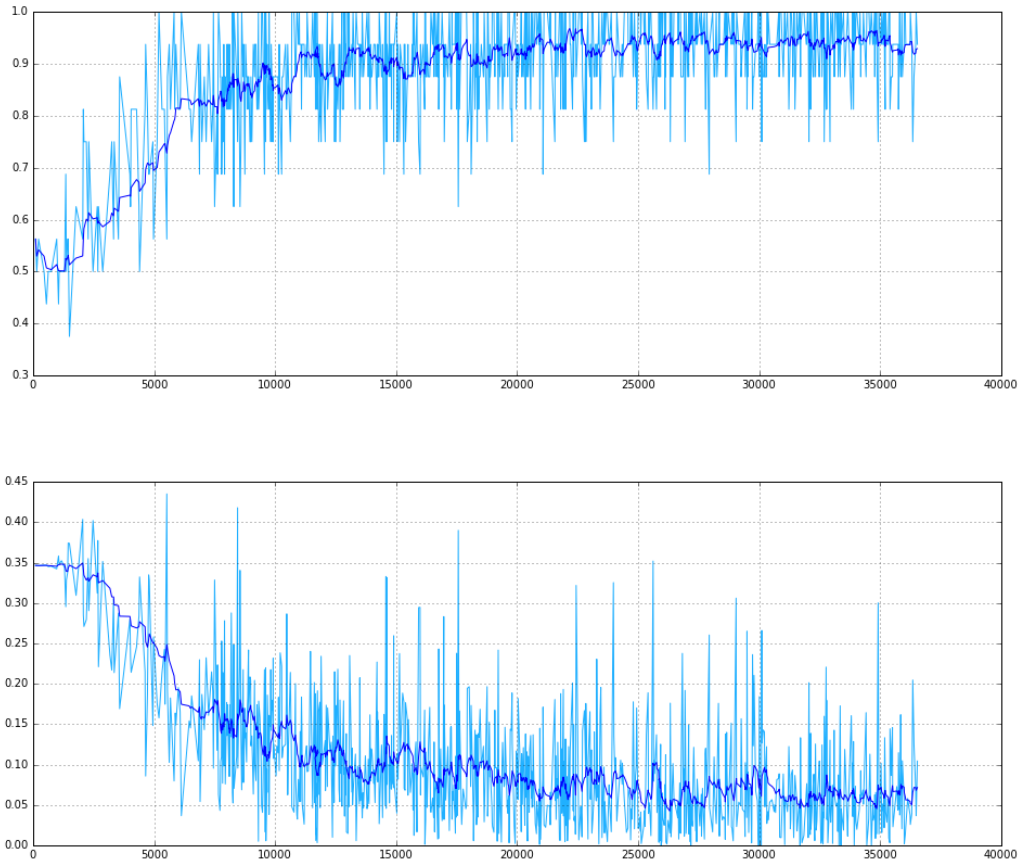


Рис. 6: Графики точности и кросс-энтропии при обучении на наборе *Kingstreet*

на более глубоких слоях можно увидеть, что подавляющее большинство начальных шумных изображений не изменилось, то есть градиент в них все время был нулевой. Это может служить свидетельством недообученности сети. После не очень большого количества итераций обучения сеть выдавала правильный результат классификации практически с нулевой функцией ошибки, что также означает, что и все градиенты по параметрам обучения тоже приблизительно равны нулю, то есть дальнейшего обучения не происходит.

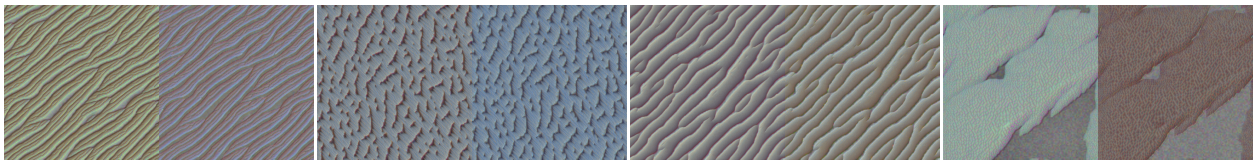
В то же время у двух сетей, обученных на более сложных данных и с использованием описанной ранее идеи для входных данных, практически все изображения получились разумными.

На удивление интересными оказались изображения фильтров первого слоя. Обычно даже при случайной инициализации матриц фильтров

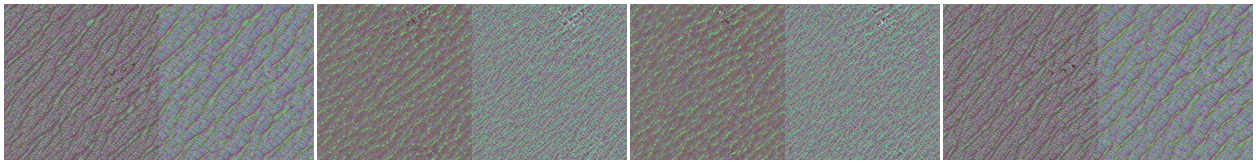
на первом слое изображения получаются не случайными, чаще всего на них изображены линии. В данном же случае практически все изображения получились монотонными, с противоположными цветами для первого и второго изображения из пары. Это говорит о том, что в первом сверточном слое нейронная сеть считает разности между первым и вторым изображениями.

С другой стороны, на последних слоях сети хотелось увидеть не абстрактные узоры, а нечто напоминающее объекты, в действительности важные для сети (такие как стрелки для *Arrows* и автомобили/пешеходы для *Kingstreet*). Более того, все изображения последних слоев очень похожи меду собой. Из этих фактов вполне может следовать то, что сеть обучилась не на распознавание каких-то реальных визуальных признаков, а были найдены какие-то закономерности во входных данных, которые могут не подойти, если данные незначительно изменить.

(a) Первый слой



(b) Второй слой



(c) Четвертый слой

Рис. 7: Некоторые визуализации внутренних нейронов классификатора *Arrows*.

6.4. Максимизация целевой функции по входному изображению

Так же, как происходила генерация изображений для вывода информации о внутреннем состоянии сети, в паре входных изображений можно зафиксировать первое изображение пары и максимизировать основную целевую функцию по второму изображению. Подав реальное входное изображение на оба входа сети и зафиксировав первое изображение и класс пары +1, можно ожидать, что, изменяя второе изображение для улучшения степени принадлежности пары нужному классу, получится "предсказать" второе изображение.

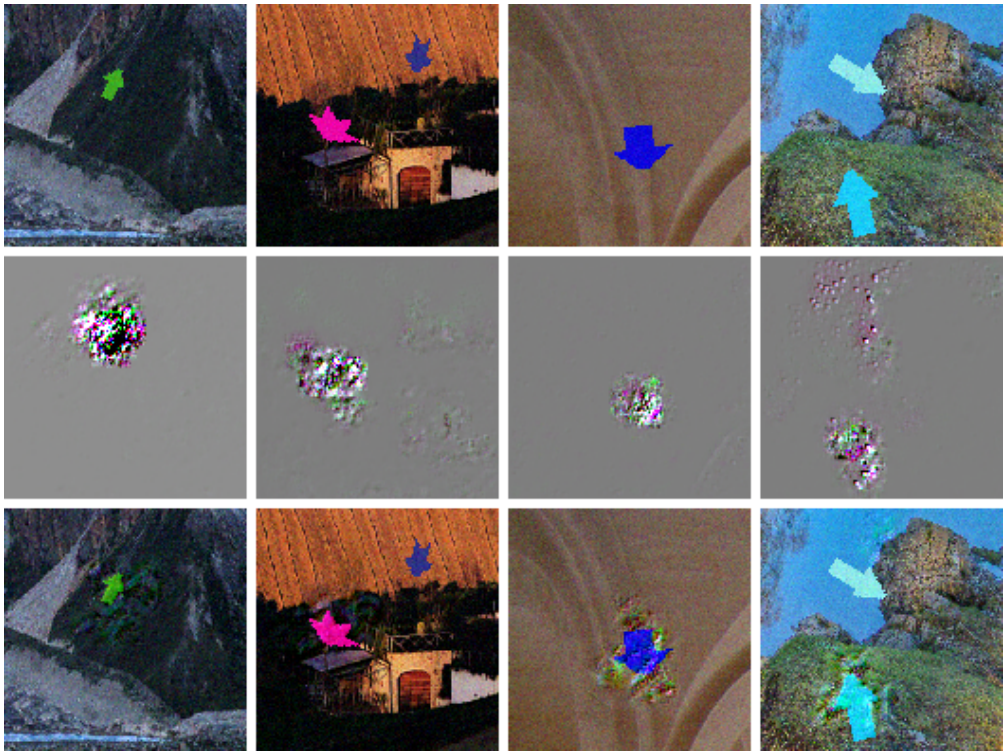


Рис. 8: Примеры градиентов изображения. Первая строка - изображения, поданные на вход сети (каждый раз на вход подается пара из двух одинаковых изображений). Вторая строка - карта значений градиентов целевой функции для изображения. Третья строка - изображения после 300 итераций применения градиентного подъема (на данном количестве итераций градиенты становятся нулевыми и изображение стабилизируется.)

В действительности, ничего похожего на следующее изображение не получается. Это логично - у сети довольно большая степень свободы;

возможно, обучив такую же сеть на огромном наборе данных, такой способ мог бы и сработать.

Тем не менее, большие по модулю градиенты появляются в "правильных" местах изображения - в районе стрелок и автомобилей. Причем для Kingstreet градиенты появляются только у машин, движущихся по дороге; машины на стоянке остаются нетронутыми. Однако эти градиенты лишь добавляют шум в окрестности этих объектов.

Рис. 9: Локализация больших значений градиента вокруг движущегося автомобиля.

7. Построение и обучение генеративной сети

Так как в работе [2] было показано, что с задачей генерации изображений хорошо справляются состязательные сети (далее GAN), и из-за схожести задачи с данной было решено применить похожий метод.

7.1. Схема генеративной сети

В работе были опробованы две разные модели GAN. В обоих случаях на стадии обучения сеть имеет в распоряжении пару $(img1, img2)$ (если быть точнее, поднабор таких пар). Для генерации напрямую используется только $img1$, $img2$ влияет лишь косвенно на моменте обучения. Наиболее подробно будет разобрана вторая модель, ибо первая не показала хороших результатов.

В корне первой модели стояла идея того, чтобы использовать обученную ранее сверточную классифицирующую сеть. Схема работы сети:

1. На вход дискриминатору подается одно изображение. Задача дискриминатора - изображениям из выборки присваивать класс $real$, а изображениям из генератора - класс $fake$. При этом параметры сети дискриминатора обучались таким образом, чтобы минимизировать функцию

$$L_D(img1, img2) = H(D(G(img1)), fake) + H(D(img2), real)$$

2. Генератору на вход подается первое изображение из пары (а не случайный вектор). Минимизируемая функция генератора:

$$L_G(img1) = H(D(G(img1)), real) + H(C(img1, G(img1), +1)$$

Вскоре от этой идеи было решено отказаться, так как старая сеть не достаточно хорошо описывает нужные классы (как можно было также убедиться в предыдущей секции в анализе сети).

На новой модели сети остановимся более подробно.

В отличие от первой модели, здесь дискриминатор принимает на входной слой пару изображений, каждый раз это либо пара $(img1, img2)$, либо $(img1, G(img1))$. Сначала изображения просто соединялись по размерности цветов, формируя шестиканальное изображение (также, как было в классифицирующей сети). Позднее было решено брать попиксельную разницу между данными двумя изображениями, чтобы уменьшить количество информации, получаемой сетью. Задача дискриминатора выставить этим парам соответствующие метки. Структура дискриминатора: обычная сверточная нейронная сеть, по архитектуре напоминающая ту сеть, которая использовалась в самом начале. Но в отличие от ранее построенного классификатора, здесь было решено сделать выходом сети не два значения - коэффициентов класса, а одно значение, к которому применялась сигмоидальная функция. Также вместо local response normalization использовалась batch-нормализация.

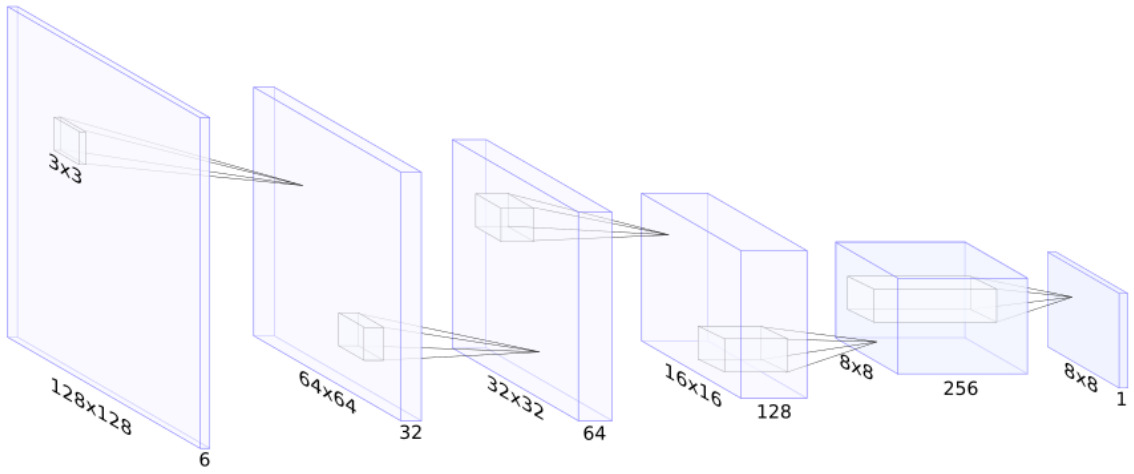
Схема генератора сложнее, чем у дискриминатора. Первая половина слоев представляет собой обычную сверточную сеть - несколько сверточных и pooling слоев с batch-нормализацией. Вторая половина сети - развертночные слои. На вход каждому развертночному слою помимо результата предыдущего слоя также подается результат свертки в ранних слоях такого же размера, чтобы сохранить больше информации.

В целевую функцию генератора входят три компоненты:

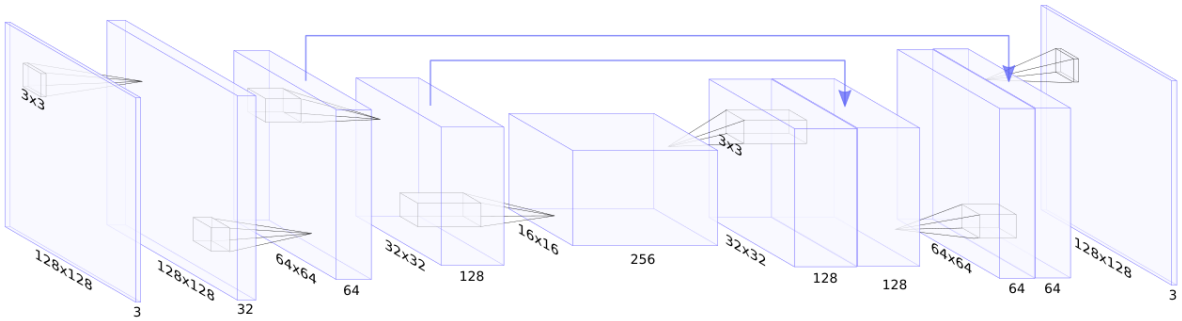
- Ошибка дискриминатора: $H(D(img1, G(img1)), real)$.
- Расстояние до первого изображения: $L1(img1, G(img1))$. Предполагается, что изменения на изображении произойдут в ограниченном количестве областей, поэтому используется метрика $L1$, штрафующая изменения как таковые и не очень сильно штрафующая крупные изменения конкретных значений.

- Расстояние до второго изображения: $L2(img2, G(img1))$. Изначально не хотелось, чтобы генератор как-то зависел от вторых изображений, но без введения этой метрики обучение сети не было замечено. Здесь используется метрика $L2$, так как хотим получить изображение, максимально похожее на второе изображение пары.

В итоге ошибкой генератора являлась сумма перечисленных ошибок с различными коэффициентами.



(a) Дискриминатор



(b) Генератор

Рис. 10: Схема GAN для *Arrows*.

7.2. Обучение сети и результаты

Основной процесс подбора правильного метода обучения происходил на наборе *Arrows*.

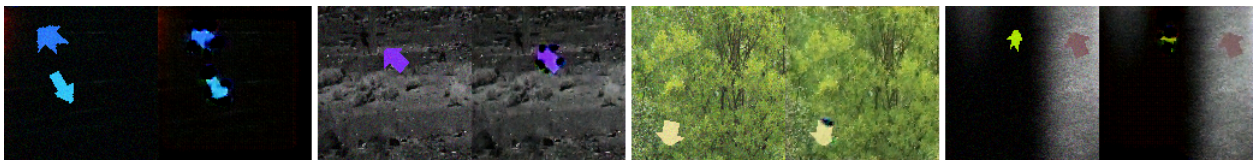


Рис. 11: Первоначальные изображения, полученные при помощи ошибок $L1$ и дискриминатора (левое изображение в паре - исходное, правое - сгенерированное)

Рис. 12: Выделение границ на ранних итерациях

Во время процесса обучения коэффициенты ошибок в целевой функции генератора часто менялись. По этой же причине не приводятся графики целевой функции - из-за того, что сама функция постоянно изменялась и сам график имеет мало смысла; процесс обучения отслеживался по получаемым изображениям генератора.

Изначально $L2$ ошибка отсутствовала, была только ошибка дискриминатора и $L1$ ошибка, и сеть обучилась до того момента, как стала генерировать изображения, по большей части похожие на те, что показаны на рисунке 11.

Черные пятна генерируются вследствие того, что $L1$ практически одинаково штрафует любой получаемый цвет, а ошибка дискриминатора еще недостаточно показательна, так как сам дискриминатор не обучен - нет генерируемых данных, более-менее соответствующих требуемому результату, и, следовательно, дискриминатору проще найти "нечестные" закономерности.

Так как дальше результаты генерации лучше не становились, обе ошибки ($L1$ и ошибка дискриминатора) были временно убраны и добавлена ошибка $L2$, а обучение дискриминатора было приостановлено. Таким образом сеть стала больше похожей на немного усложненный autoencoder. Результаты обучение сети можно увидеть на рисунке 12. За достаточно короткое время сеть научилась генерировать изображения, похожие на входные (разве что немного сглаженные). Через некоторое

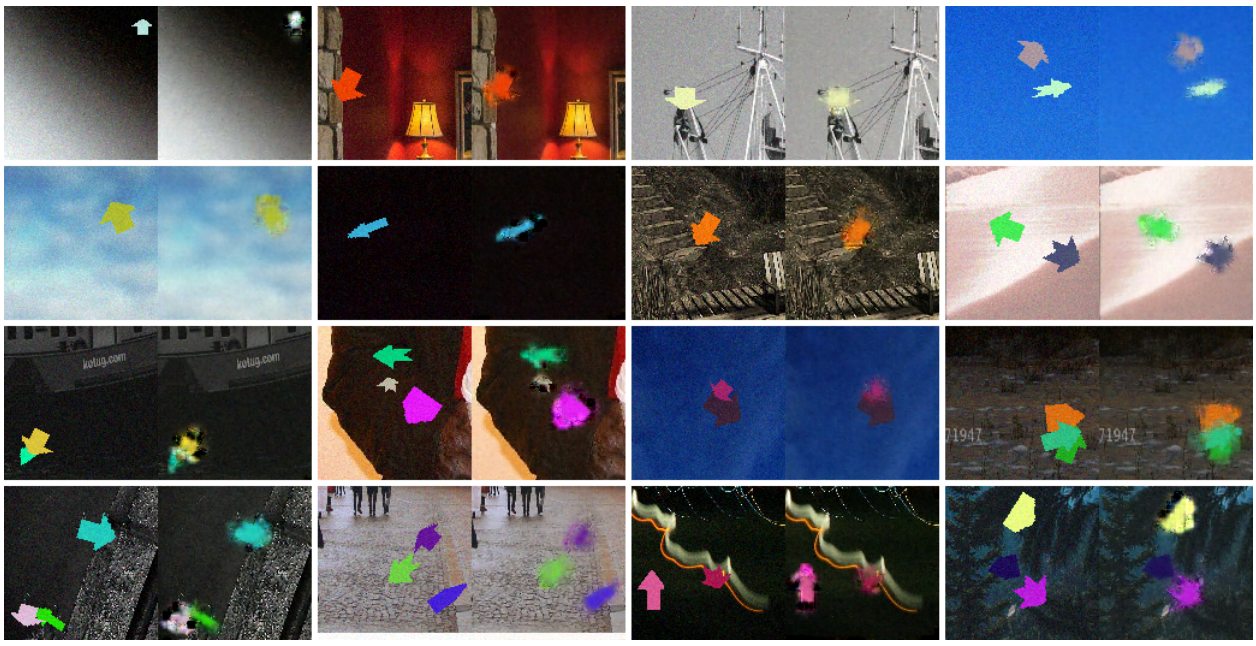


Рис. 13: Результаты работы генератора

время у стрелок стали выделяться контуры, при том, что остальные части изображения не изменялись. После чего за довольно продолжительное время были результаты, напоминающие желаемый результат - движение стрелок происходило, фон за ними заполнялся примерно той же текстурой, что и вокруг.

Но на полученных изображениях стрелки слишком сильно размыты. Это происходит из-за того, что в исходных данных они перемещаются на случайное небольшое расстояние, и в итоге генератор "двигает стрелки на все эти расстояния понемногу". В надежде, что дискриминационная сеть все-таки сможет с этим помочь, отделяя изображения с размытыми стрелками от более четких, на этом этапе была возвращена ошибка дискриминатора, однако, к сожалению, после продолжительного обучения сети никаких улучшений не последовало, с учетом множественного подбора подходящих коэффициентов для ошибки генератора.

Заключение

В работе было проведено несколько исследований.

Была успешна обучена сеть-классификатор, определяющая направление течения времени между изображениями в паре; точность классификации достигает 95%. На основе обученной сети был использован анализ ее внутренних слоев с использованием визуализации изображений, максимизирующих вывод внутренних нейронов сети. При визуализации были отмечены различные свойства сети: наличие подсчета дельт изображений на первом слое, недообученность первой версии сети, а также было обращено внимание на однообразность изображений последних слоев.

В работе был построен генератор изображения. Для этого были использованы следующие методы: аналог сети-автоэнкодера, состоящей из последовательных сверточных и разверточных слоев, и генеративная состязательная сеть, характеризующаяся наличием дискриминационной сети. Сравнение методов показало, что более простой метод с автоэнкодером, без дискриминационной сети, лучше подошел для поставленной задачи, не смотря на то, что состязательные сети хорошо показали себя в другой подобной задаче.

Исходный код доступен по ссылке: `https://github.com/vlpolyansky/video-cnn`

Список литературы

[1] Deconvolutional networks / M. D. Zeiler, D. Krishnan, G. W. Taylor, R. Fergus // Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on. — 2010. — June. — P. 2528–2535.

[2] FlowNet: Learning Optical Flow with Convolutional Networks / Philipp Fischer, Alexey Dosovitskiy, Eddy Ilg et al. // CoRR. — 2015. — Vol. abs/1504.06852. — URL: http://arxiv.org/abs/1504.06852.

[3] Generative Adversarial Nets / Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza et al. // Advances in Neural Information Processing Systems 27 / Ed. by Z. Ghahramani, M. Welling, C. Cortes et al. — Curran Associates, Inc., 2014. — P. 2672–2680. — URL: http://papers.nips.cc/paper/5423-generative-adversarial-nets.pdf.

[4] Ioffe Sergey, Szegedy Christian. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift // CoRR. — 2015. — Vol. abs/1502.03167. — URL: http://arxiv.org/abs/1502.03167.

[5] Krizhevsky Alex, Sutskever Ilya, Hinton Geoffrey E. ImageNet Classification with Deep Convolutional Neural Networks // Advances in Neural Information Processing Systems 25 / Ed. by F. Pereira, C. J. C. Burges, L. Bottou, K. Q. Weinberger. — Curran Associates, Inc., 2012. — P. 1097–1105. — URL: http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-netw pdf.

[6] Large-scale Video Classification with Convolutional Neural Networks / Andrej Karpathy, George Toderici, Sanketh Shetty et al. // CVPR. — 2014.

[7] LeCun Yann, Kavukvuoglu Koray, Farabet Clément. Convolutional

Networks and Applications in Vision // Proc. International Symposium on Circuits and Systems (ISCAS'10). — IEEE, 2010.

[8] Radford Alec, Metz Luke, Chintala Soumith. Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks // CoRR. — 2015. — Vol. abs/1511.06434. — URL: http://arxiv.org/abs/1511.06434.

[9] Xie Junyuan, Girshick Ross, Farhadi Ali. Deep3D: Fully Automatic 2D-to-3D Video Conversion with Deep Convolutional Neural Networks. — URL: https://homes.cs.washington.edu/~jxie/pdf/deep3d.pdf.

[10] Zeiler Matthew D., Fergus Rob. Visualizing and Understanding Convolutional Networks // CoRR. — 2013. — Vol. abs/1311.2901. — URL: http://arxiv.org/abs/1311.2901.