Санкт-Петербургский государственный университет

Выпускная квалификационная работа

общей образовательной программы бакалавриата
«Прикладная математика и информатика»

студентки 4 курса, группы 17.Б05-мм
Анастасии Ивановны Волошиной

на тему

«Развитие методов обучения искусственных нейронных сетей для задач компьютерного зрения»

Научный руководитель:
к.ф.-м.н., Михаил Сергеевич Ананьевский
доцент кафедры теоретической кибернетики

Рецензент:
д.т.н., Игорь Борисович Фуртат
в.н.с. лаб. УСС ИПМаш РАН

г. Санкт-Петербург
2021

Saint Petersburg State University

Graduation Project

Applied Mathematics and Computer Science
Control and Processing of Information in Cybernetical and Robotic Systems

Anastasiia Voloshina

Development of methods for training artificial neural networks for computer vision problems

Scientific Supervisor:
Mikhail Ananyevskiy
Candidate of Physico-Mathematical Sciences

Reviewer:
Igor Furtat
Doctor of Engineering Sciences

Saint Petersburg
2021

Содержание

1 Введение

В задачах машинного обучения качество моделей очень сильно зависит от данных. Но сами данные в реальных задачах редко бывают идеальными. Распространены следующие проблемы:

1. Малый размер набора данных.
2. Недостаток данных определенного типа [1], то есть ситуация, при которой модель не может обучиться качественно работать с некоторым признаком.
3. Несбалансированное обучающее множество [2], в котором доля примеров некоторого класса слишком мала.
4. Ложные корреляции исходных данных [3], когда не зависящие друг от друга признаки имеют схожее поведение. Это может подтолкнуть к ложным выводам о наличии причинно-следственной связи между явлениями.
5. Нерепрезентативность выборочных данных [4], то есть случай, при которой обучающая выборка не отображает свойств генеральной совокупности.
6. Различные условия сбора данных для обучения и дальнейшего применения модели. Характерный пример — обучение и применение модели на двух датасетах с изображениями людей, но сформированных различным образом.
7. Ограниченность доступных для анализа параметров.
8. Присутствие шумов в наборе данных.

Использование таких данных при моделировании может приводить к неверным результатам. Поэтому важным этапом работы с данными является их предварительная обработка.

В 1998 году был представлен набор данных MNIST [5] — маркированный набор изображений рукописного написания цифр. Наицональным институтом стандартов и технологий США было предложено использовать этот набор в качестве стандарта для сопоставления методов распознавания изображений с помощью машинного обучения. Распознавание рукописных цифр — сложная проблема, которая интенсивно изучалась в течение многих лет в области распознавания рукописного текста. Многочисленные результаты были достигнуты исследователями, которые использовали различные алгоритмы, такие как

- K-ближайшие соседи (KNNs) [6]
- машины опорных векторов (SVMs) [6]
- нейронные сети (NNs) [7]
- сверточные нейронные сети (CNNs) [8]

Задача распознавания рукописных цифр является важной во многих приложениях, включая автоматизированную сортировку почты по почтовому коду, автоматизированное чтение чеков и налоговых деклараций, а также ввод данных для портативных компьютеров. В этой области достигнут быстрый прогресс. Один из наилучших результатов точности распознавания 99,65% на датасете MNIST был достигнут с помощью 6-слойной свёрточной нейронной сети с предварительно обработанными данными [8].

Одного только усложнения архитектуры моделей CNN недостаточно для достижения лучших результатов точности классификации для любого набора данных. Методы предварительной обработки играют жизненно важную роль в достижении уровня техники в любом наборе данных. В данной работе рассмотрен метод обеливания данных, который является распространенным этапом предварительной обработки в статистическом анализе для преобразования случайных величин в ортогональные. Однако, как будет показано далее, благодаря вращательной свободе поворотов в пространстве признаков существует бесконечно много возможных процедур обеливания.

1.1 Постановка задачи

В данной работе подробно рассмотрено линейное преобразование обеливания, а также его варианты методом анализа главных компонент (PCA) и методом анализа нулевых компонент (ZCA). В том числе исследовалось влияние предварительной обработки данных методом обеливания на точность распознавания в задаче компьютерного зрения. В качестве данных взят классический набор рукописных цифр MNIST.

2 Обеливание

Обеливание (англ. whitening) или сферирование (англ. sphering) — это линейное преобразование, используемое для декорреляции сигналов. Оно является распространенным этапом предварительной обработки в статистическом анализе для преобразования случайных величин в ортогональные. Термин «обеливание» происходит от белого шума (который, в свою очередь, получил свое название от белого света), состоящего из последовательно некоррелированных образцов. Таким образом, с помощью обеливания можно получить вектор белого шума с некоррелированными компонентами из случайного начального вектора. Однако благодаря вращательной свободе преобразования существует бесконечно много возможных процедур обеливания.

Обеливание преобразует случайный $d-$мерный вектор $x = (x_1, \ldots, x_d)^T$ со средним $\mathbb{E}(x) = \mu = (\mu_1, \ldots, \mu_d)^T$ и положительно определённой ковариационной $d \times d$ матрицей $var(x) = \Sigma$ в новый случайный вектор

$$z = (z_1, \ldots, z_d)^T = Wx \tag{1}$$

той же размерности d с единичной диагональной ковариационной матрицей $var(z) = I$. Квадратная $d \times d$ матрица W называется матрицей обеливания. Поскольку ортогональность случайных величин значительно упрощает анализ многомерных данных как с вычислительной, так и со статистической точки зрения, обеливание является критически важным инструментом, чаще всего используемым в предварительной обработке.

Обеливание можно рассматривать как обобщение стандартизации случайной величины, которое осуществляется как

$$z = V^{-1/2}x, \tag{2}$$

где матрица $V = diag(\sigma_1^2, \ldots, \sigma_d^2)$ содержит дисперсии $var(x_i) = \sigma_i^2$. Это приводит к $var(z_i) = 1$, но не устраняет корреляции. Часто стандартизация и обеливающие преобразования также сопровождаются центрированием среднего значения x или z для обеспечения $\mathbb{E}(z) = 0$, но на самом деле это не является

необходимым для получения единичных дисперсий или единичной матрицы ковариации.

Для преобразования обеливания, определенного в уравнении (1), необходим выбор подходящей матрицы обеливания W. Из того, что $var(z) = I$ следует

$$W\Sigma W^T = I$$

и $W(\Sigma W^T W) = W$, что выполняется, если W удовлетворяет условию

$$WW^T = \Sigma^{-1}. \tag{3}$$

К сожалению, это ограничение не однозначно определяет матрицу обеливания W. Существует бесконечно много возможных матриц W, соответствующих Σ, каждая из которых удовлетворяет (3) и приводит к преобразованию обеливания, которое создаёт ортогональные, но различные сферические случайные величины. Следовательно, существует широкий спектр используемых методов, например, основанных на анализе главных компонент (PCA), разложении матрицы Холески и анализе нулевых компонент (ZCA) и других.

2.1 Свобода вращения при обеливании

Рассмотрим ряд тождеств ковариационной матрицы, которые будут использоваться далее. Разложение ковариационной матрицы на корреляционную матрицу P и диагональную дисперсионную матрицу V запишется как

$$\Sigma = V^{1/2} P V^{1/2},$$

а собственное разложение ковариационной матрицы выглядит так:

$$\Sigma = U \Lambda U^T,$$

где U содержит собственные векторы, а Λ — собственные значения Σ. Также будет использован уникальный квадратный корень обратной матрицы $\Sigma^{-1/2} = U \Lambda^{-1/2} U^T$.

Аналогично для корреляционной матрицы P собственное разложение записывается в виде

$$P = G\Theta G^T,$$

где G содержит собственные векторы и Θ собственные значения. А также $P^{-1/2} = G\Theta^{-1/2}G^T$, уникальный квадратный корень обратной матрицы корреляционной матрицы.

Предполагается, что собственные значения сортируются в порядке от наибольшего к наименьшему значению, а собственные векторы при построении определяются только до знака, то есть столбцы U и G могут быть умножены с коэффициентом -1, и полученная матрица остается действительной.

Свобода вращения преобразования обеливания описана в статье «Optimal Whitening and Decorrelation» [9]. Уравнение (3) не полностью определяет W, но допускает свободу вращения. Это становится очевидным, если записать W в полярном разложении

$$W = Q_1\Sigma^{-1/2}, \tag{4}$$

где Q_1 ортогональная матрица такая, что $Q_1^TQ_1 = I_d$. Очевидно, W удовлетворяет (3) независимо от Q_1.

Геометрически (4) интерпретируется как комбинации многомерного масштабирования матрицей $\Sigma^{-1/2}$ и вращения на Q_1. Оно также показывает, что все матрицы обеливания W имеют одинаковые сингулярные значения $\Lambda^{-1/2}$, что следует из сингулярного разложения $W = (Q_1U)\Lambda^{-1/2}U^T$ с ортогональностью Q_1U. Это подчеркивает, что фундаментальное масштабирование осуществляется через квадратный корень из собственных значений $\Lambda^{-1/2}$. Геометрически преобразование обеливания с $W = Q_1U\Lambda^{-1/2}U^T$ представляет собой вращение U^T с последующим масштабированием, и возможно, с последующим другим вращением (в зависимости от выбора Q_1).

Поскольку во многих ситуациях желательно работать со стандартизированными переменными $V^{-1/2}x$, еще одним полезным разложением W, которое также непосредственно демонстрирует присущую ему вращательную свободу, яв-

ляется

$$W = Q_2 P^{-1} V^{-1}, \tag{5}$$

где Q_2 — еще одна ортогональная матрица, такая что $Q_2^T Q_2 = I_d$. Очевидно, что такое W также удовлетворяет ограничению, которое накладывает уравнение (3), независимо от выбора Q_2.

В этом представлении при $W = Q_2 G \Theta^{-1/2} G^T V^{-1/2}$ переменные сначала масштабируются квадратным корнем диагональной дисперсионной матрицы, затем поворачиваются на G^T, затем снова масштабируются квадратным корнем собственных значений корреляционной матрицы и, возможно, поворачиваются еще раз (в зависимости от выбора Q_2). Для того чтобы вышеприведенные два представления привели к одной и той же матрице обеливания W, требуются два различных поворота Q_1 и Q_2, которые связаны соотношением $Q_1 = Q_2 A$, где матрица $A = P^{-1/2} V^{-1/2} \Sigma^{1/2} = P^{1/2} V^{1/2} \Sigma^{-1/2}$ сама ортогональна [9]. Поскольку представления через композицию собственных векторов и значений матрицы ковариации и корреляционной матрицы не так просто связаны друг с другом, матрица A не может быть упрощена.

2.2 PCA обеливание

«PCA-whitening» основано на масштабированном анализе главных компонент (англ. principal component analysis, PCA) и использует матрицу обеливания

$$W^{PCA} = \Lambda^{-1/2} U^T. \tag{6}$$

Такое преобразование сначала вращает переменные, используя собственную матрицу ковариации Σ, как это делается в стандартном PCA. Это приводит к ортогональным компонентам, но с различными дисперсиями. Для получения обеленных данных повернутые переменные затем масштабируются на квадратный корень из собственных значений $\Lambda^{-1/2}$.

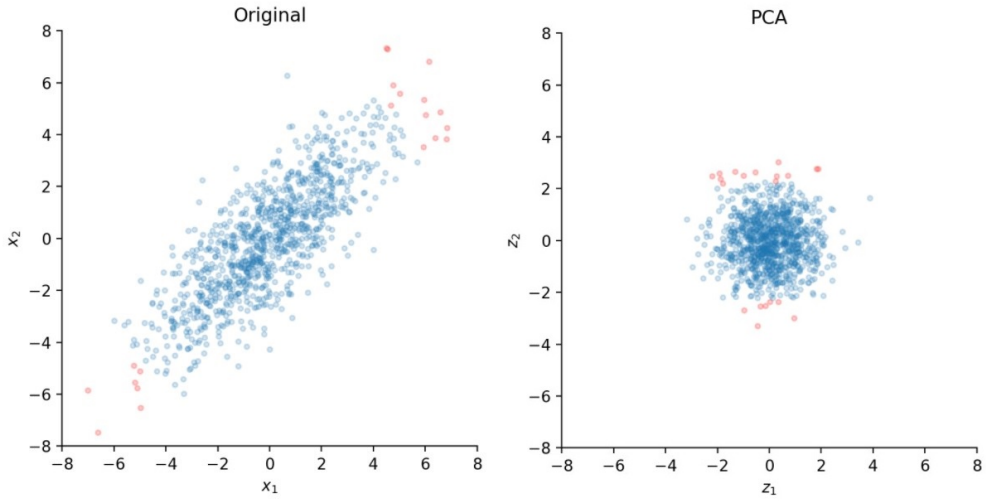

Рис. 1: Визуализация применения обеливания методом главных компонент на сгенерированных данных 1000 выборок двух коррелированных временных рядов x_1 и x_2 [10]. Красным отмечены 20 самых крайних значений.

2.3 ZCA обеливание

«ZCA-whitening» — это популярный метод нормализации ввода, похожий на стандартную нормализацию и «PCA-whitening», но разработанный специально для изображений, видео и других естественно упорядоченных типов данных. Впервые к обучающим данным этот метод был применен в работе [11]. Есть основания предполагать, что предварительная обработка данных изображений с помощью ZCA-обеливания обеспечивает наилучшую производительность для свёрточной нейронной сети [12].

Преобразование обеливания ZCA, также известное как обеливание Махаланобиса, основанно на методе анализа нулевых компонент (англ. zero-phase component analysis, ZCA) и использует матрицу обеливания

$$W^{ZCA} = \Sigma^{-1/2}. \tag{7}$$

При $Q1 = I$ это уникальный метод сферирования с симметричной обеливающей матрицей.

Из рис. 2 следует, что преобразования PCA- и ZCA-обеливания связаны вращением U, поэтому ZCA-обеливание можно интерпретировать как вращение с

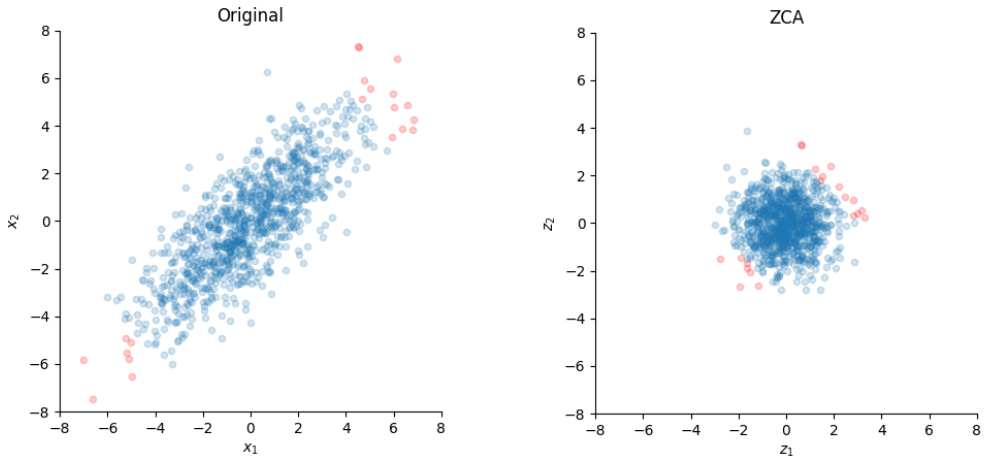

Рис. 2: Визуализация применения обеливания методом анализа нулевых компонент на сгенерированных данных 1000 выборок двух коррелированных временных рядов x_1 и x_2 [10]. Красным отмечены 20 самых крайних значений.

последующим масштабированием, за которым следует вращение U обратно в исходную систему координат. При $Q_1 = I$ для ZCA-сферирования и $Q_1 = U^T$ для PCA-сферирования матрицы обеливания удовлетворяют полярному разложению из уравнения (4).

3 Свёрточные нейронные сети

Сверточная нейронная сеть (СНС) была впервые представлена в работе [13]. Это специальный вид нейронной сети для обработки данных с сеточной топологией. Примерами могут служить временные ряды, которые можно рассматривать как одномерную сетку наблюдений, выбираемых через регулярные промежутки времени, а также изображения, рассматриваемые как двумерная сетка пикселей. Названы они так в силу использования математической операции свертки — особого вида линейной операции, которая преобразует входное изображение в карту признаков.

Свёрточный слой (англ. convolutional layer) — основной блок СНС, включающий в себя для каждого канала изображения свой фильтр. Ядро свёртки фильтра обрабатывает предыдущий слой, суммируя результаты поэлементного произведения для каждого фрагмента. Веса ядер, то есть коэффициенты матриц весов, настраиваются в процессе обучения.

Слой пулинга (агнл. pooling layer) — слой подвыборки, представляющий собой нелинейное уплотнение карты признаков, при этом группа пикселей уплотняется до одного, проходя нелинейное преобразование ядром пулинга. Существует два наиболее распространенных типа: максимальный и средний. Максимальный пулинг использует максимальное значение каждого локального кластера нейронов, сотвествующего размеру ядра, в карте признаков, в то время как средний пулинг принимает среднее значение.

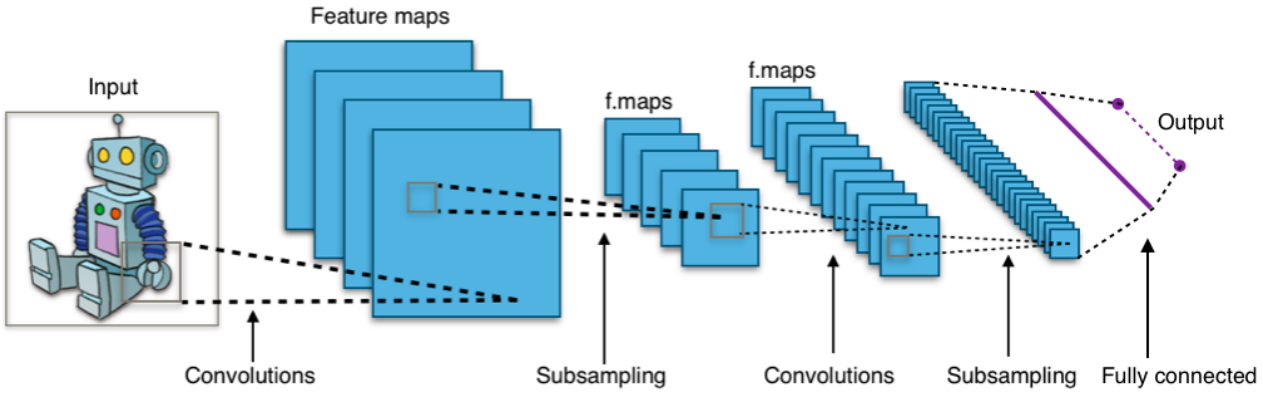

Рис. 3: Типовая архитектура свёрточной нейронной сети [14].

У естественных изображений есть много статистических свойств, инвариантных относительно параллельного переноса. Например, фотография кошки остается таковой, если сдвинуть ее на один пиксель вправо. В СНС это свойство учитывается с помощью разделения параметров по нескольким областям изображения, то есть наложения некоторых ограничений, требующих, чтобы множества параметров совпадали. Один и тот же признак (скрытый блок с одинаковыми весами) вычисляется по нескольким участкам входных данных. Это означает, что один и тот же детектор кошек найдет кошку вне зависимости от того, находится она в столбце изображения с номером i или $i + 1$. Разделение параметров дает СНС возможность значительно уменьшить число уникальных параметров модели и увеличить размер сети, не требуя соответственного увеличения объема обучающих данных.

3.1 Реализация СНС

В ходе работы для написания СНС на языке Python использовалась открытая библиотека TensorFlow [15]. Модель обучалась для распознавания цифр из стандартного набора — MNIST, который содержит 70 000 черно-белых изображений. На изображениях показаны отдельные цифры с низким разрешением (28 на 28 пикселей). Метки — это массив целых чисел от 0 до 9, соответствующих цифре. Для обучения использовались 60 000 изображений, остальные — для тренировки. Значения пикселей масштабировались до диапазона $[0, 1]$.

3.1.1 Первая архитектура

Входной слой является свёрточным с ядром 3×3 *и функцией активации ReLu* $f(x) = \max(0, x)$. *Далее идут скрытые слои:*

- *слой пулинга с ядром* 2×2,
- *свёрточный слой с ядром* 3×3 *и функцией активации ReLu,*
- *слой пулинга с ядром* 2×2,

- *полносвязный нейронный слой, который содержит 64 узла (нейрона).*

Выходной слой — полносвязный нейронный слой из 10 узлов и функция активации «Softmax»

$$\sigma_i(x) = \frac{e^{x_i}}{\sum\limits_{k=1}^{10} e^{x_k}}.$$

Выходной слой возвращает массив из 10 вероятностных оценок. Функция «Softmax» принимает в качестве входных данных вектор из 10 действительных чисел и нормализует его в распределение вероятностей, состоящее из 10 вероятностей, пропорциональных экспонентам входных чисел. Сумма компонент составит 1, так что их можно интерпретировать как вероятности принадлежности полученного на вход изображения к одному из классов.

3.1.2 Вторая архитектура

Второй вариант СНС имеет похожую, но более простую структуру. Входной слой также является свёрточным с ядром 3×3 *и функцией активации ReLu* $f(x) = \max(0, x)$. *Далее идут скрытые слои:*

- *слой пулинга с ядром* 2×2*,*
- *полносвязный нейронный слой, который содержит 64 узла (нейрона).*

За ними следует выходной слой — полносвязный нейронный слой из 10 узлов и функция активации «Softmax».

Для измерения точности моделей во время обучения минимизировалась функция потерь из стандартной библиотеки Keras — categorical crossentropy:

$$-\sum_{i=1}^{K} y_i \cdot log(\sigma_i),$$

где y_i — целевой $i-$ый класс, σ_i — вероятность принадлежности $i-$му классу, вычисленная функцией активации Softmax, K — количество классов. Для обновления моделей во время обучения использовался метод стохастической оптимизации — Adam [16].

Весь код представлен в разделе «Приложение».

4 Результаты

Для сравнения влияния предобработки данных на точность распознавания обученной нейронной сети были созданы три модели с идентичной структурой. Первая модель являлась эталонной, так как обучалась распознаванию на исходных данных. Вторая и третья модели обучались на обеленных данных методом PCA и ZCA соответсвенно.

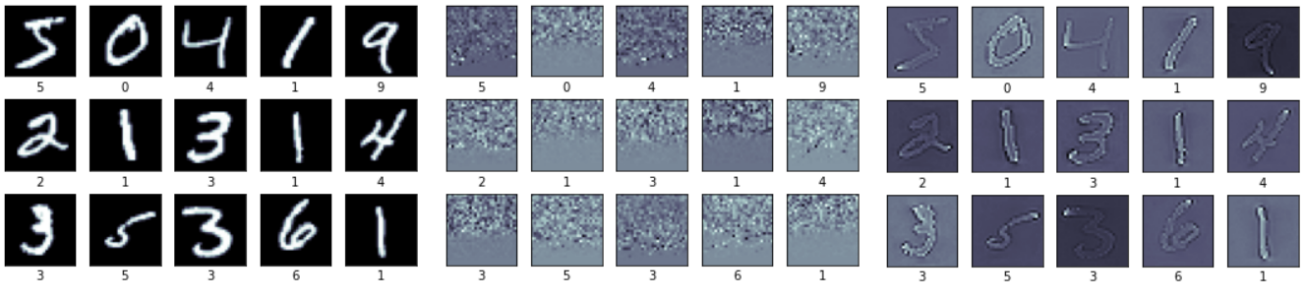

Рис. 4: Первые 15 изображений из обучающего набора: исходные данные слева, обеленные методом PCA по центру и обеленные методом ZCA справа.

На рисунке 4 видно, что PCA-обеливание полностью разрушает пространственную структуру исходных изображений — ни одно из исходных чисел не видно в этих преобразованных версиях, в отличие от ZCA-обеливания. В связи с этим отпадает необходимость использовать свёрточную нейронную сеть для обучения и дальнейшего распознавания, поскольку нет паттернов на изображении (линии, формы, ориентированные края, углы). Подобный вывод можено сделать и из графиков точности при обучении на PCA-обеленных данных и данных, которые преобразованы случайной матрицей. Сравнение отображено на рис. 5.

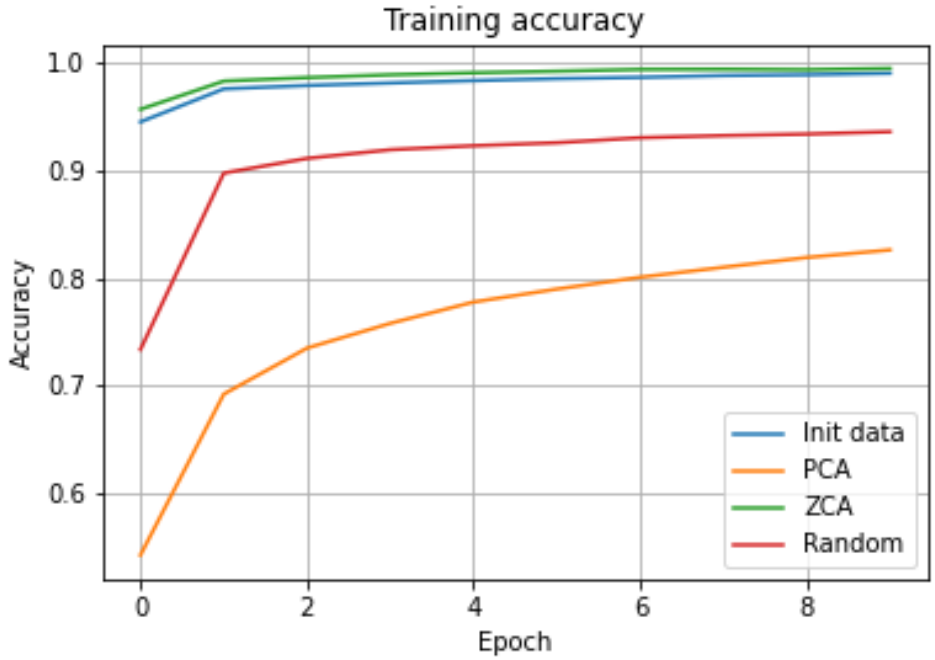

Рис. 5: Точность при обучении в течение 10 эпох 1-й модели СНС на различных данных: «Init data» — исходных данных из набора MNIST, «PCA» — обеленных с помощью метода PCA, «ZCA» — обеленных методом ZCA, «Random» — данных, преобразованных случайной матрицей.

Количественные результаты распознавания моделями первой архитектуры представлены в таблице 1.

Таблица 1. Результаты моделей с 1-й архитектурой

	Точность при обучении	Тестовая точность
Исходные данные	0.9905	0.9861
PCA-обеленные	0.8295	0.7801
ZCA-обеленные	0.9898	0.9853
Случайно преобразованные	0.9378	0.9280

Даже для случайно преобразованных данных обученная на них модель 1-й архитектуры показала достаточно хорошую точность распознавания 92%. Поэтому были созданы модели более простой архитектуры свёрточной нейронной сети.

Таблица 2. Результаты моделей со 2-й архитектурой

	Точность при обучении	Тестовая точность
Исходные данные	0.9902	0.9726
PCA-обеленные	0.9369	0.8885
ZCA-обеленные	0.9913	0.9710
Случайно преобразованные	0.9861	0.9620

Как видно по данным таблицы 2, в которой представлены результаты точности распознавания моделей второй архитектуры, нейронная сеть незначительно переобучается. На рис. 6 это заметно по приросту в течение последних 5 эпох обучения. Точность незначительно увеличивается долгое время обучения, при этом точность распознавания на тестовых данных значительно меньше.

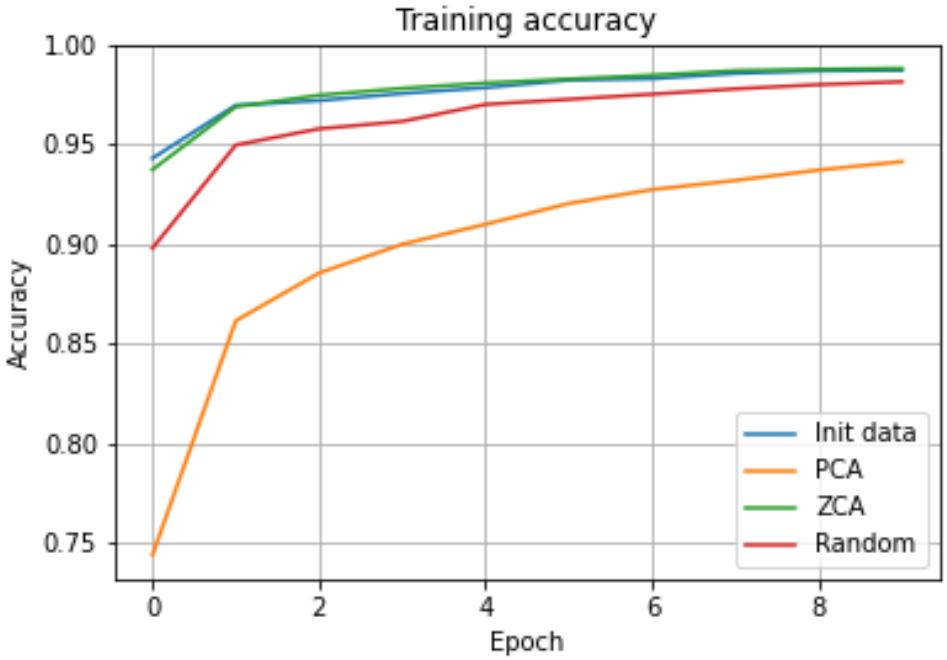

Рис. 6: Точность при обучении в течение 10 эпох 2-й модели СНС на различных данных: «Init data» — исходных данных из набора MNIST, «PCA» — обеленных с помощью метода PCA, «ZCA» — обеленных методом ZCA, «Random» — данных, преобразованных случайной матрицей.

5 Заключение

В данной работе было рассмотрено влияние предварительной обработки данных на точность распознавания изображений в задачах компьютерного зрения, а именно рассмотрено обеливание данных и его реализация методом анализа главных компонент (PCA) и методом анализа нулевых компонент (ZCA). Были реализованы модели двух архитектур свёрточной нейронной сети на языке Python. В качестве данных использовался классических набор рукописных цифр MNIST. Был проведён сравнительный анализ точности при обучении и точности на тестовых данных построенных моделей СНС на исходных и обеленных, а также случайно преобразованных данных.

Из полученнных графиков точности распознавания моделей обеих архитектур в течение 10 эпох обучения следует, что предварительная обработка изображений PCA-обеливанием значительно замедляет скорость обучения. Даже на случайно преобразованных данных обе модели обучаются лучше. Однако предварительная обработка ZCA-обеливанием, сохраняющая пространственную структуру изображений, улучшает точность распознавания и на обучающей, и на тестовой выборках.

Численные эксперименты, проведенные в рамках работы, показали, что даже при «искажении» исходных данных некоторым (неизвестным для нейронной сети) линейным преобразованием, в результате которого исходное изображение изменяется до неузнаваемости, нейронная сеть оказывается способной хорошо обучиться для распознавания классов. Это наблюдение может быть очень полезно для задач обучения нейронных сетей без разглашения обучающего датасета, что имеет большое практическое значение.

6 Список литературы

[1] Gonfalonieri Alexandre. Dealing with the Lack of Data in Machine Learning. May 17, 20195. URL: https://medium.com/predict/dealing-with-the-lack-of-data-in-machine-learning-725f2abd2b92.

[2] Кафтанников И.Л., Парасич А.В. Проблемы формирования обучающей выборки в задачах машинного обучения // Вестник ЮУрГУ. Серия «Компьютерные технологии, управление, радиоэлектроника.». 2016. Т. 16, № 3. С. 15–24.

[3] Yule G. Why do we Sometimes get Nonsense-Correlations between Time-Series?–A Study in Sampling and the Nature of Time-Series // Journal of the Royal Statistical Society. Т. 89. с. 1.

[4] The data representativeness criterion: Predicting the performance of supervised classification based on data set similarity / Evelien Schat, Rens van de Schoot, Wouter M. Kouw [и др.] // PLOS ONE. 2020. Aug. Т. 15, № 8. с. e0237009. URL: http://dx.doi.org/10.1371/journal.pone.0237009.

[5] LeCun Y., Bottou L., Bengio Y. [и др.]. THE MNIST DATABASE of handwritten digits. 1998. URL: http://yann.lecun.com/exdb/mnist/.

[6] Gradient-based learning applied to document recognition / Y. Lecun, L. Bottou, Y. Bengio [и др.] // Proceedings of the IEEE. 1998. Т. 86, № 11. С. 2278–2324.

[7] Deep, Big, Simple Neural Nets for Handwritten Digit Recognition / Dan Claudiu Cireşan, Ueli Meier, Luca Maria Gambardella [и др.] // Neural Computation. 2010. Dec. Т. 22, № 12. с. 3207–3220. URL: http://dx.doi.org/10.1162/NECO_a_00052.

[8] Flexible, High Performance Convolutional Neural Networks for Image Classification. / Dan Ciresan, Ueli Meier, Jonathan Masci [и др.] //

International Joint Conference on Artificial Intelligence IJCAI-2011. 2011. 07. С. 1237–1242.

[9] Kessy Agnan, Lewin Alex, Strimmer Korbinian. Optimal Whitening and Decorrelation // The American Statistician. 2018. Jan. Т. 72, № 4. с. 309–314. URL: http://dx.doi.org/10.1080/00031305.2016.1277159.

[10] Brunner Clemens. Whitening with PCA and ZCA. 2015. URL: https://cbrnr.github.io/posts/whitening-pca-zca/.

[11] Krizhevsky Alex. Learning Multiple Layers of Features from Tiny Images // University of Toronto. 2012. 05.

[12] Pal Kuntal, Sudeep K. Preprocessing for image classification by convolutional neural networks. 2016. 05. С. 1778–1781.

[13] Backpropagation Applied to Handwritten Zip Code Recognition / Y. LeCun, B. Boser, J. S. Denker [и др.] // Neural Computation. 1989. Т. 1, № 4. С. 541–551.

[14] Aphex34. Типовая архитектура свёрточной нейронной сети. CC BY-SA 4.0. URL: https://commons.wikimedia.org/w/index.php?curid=45679374.

[15] Abadi Martín, Agarwal Ashish, Barham Paul [и др.]. TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems. 2015. Software available from tensorflow.org. URL: https://www.tensorflow.org/.

[16] Kingma Diederik P., Ba Jimmy. Adam: A Method for Stochastic Optimization. 2017.

7 Приложение

```
"""# Data"""
import tensorflow as tf

#load dataset MNIST digits
(train_images, train_labels), (test_images, test_labels) = tf.keras.datasets.
    mnist.load_data()

#shape of subsets
print(train_images.shape)
print(test_images.shape)

import matplotlib.pyplot as plt
def image_plotting(data):
    fig, ax = plt.subplots(3, 5)
    for i, axi in enumerate(ax.flat):
        axi.imshow(data[i], cmap='bone')
        axi.set(xticks=[], yticks=[], xlabel=train_labels[i])

image_plotting(train_images[0:15])

import numpy as np
X = np.concatenate((train_images, test_images), axis=0)

# Normalize pixel values to be between 0 and 1
X = X / 255.0

"""# Whitening"""

def whiten(X, method):
    X = X.reshape((-1, np.prod(X.shape[1:])))
    X_centered = X - np.mean(X, axis=0)
    Sigma = np.dot(X_centered.T, X_centered) / X_centered.shape[0]
    W = None

    if method in ['zca', 'pca', 'random']:
        U, Lambda, _ = np.linalg.svd(Sigma)
        if method == 'zca':
            W = np.dot(U, np.dot(np.diag(1.0 / np.sqrt(Lambda + 1e-6)), U.T))
        elif method =='pca':
            W = np.dot(np.diag(1.0 / np.sqrt(Lambda + 1e-6)), U.T)
        elif method == 'random':
```

```
            W = np.random.rand(784, 784)
    else:
        raise Exception('Whitening method not found.')

    return np.dot(X_centered, np.transpose(W))

"""## With PCA """
PCA_DATA = whiten(X, method='pca')
image_plotting(PCA_DATA[0:15].reshape(15,28,28))

"""## With ZCA"""
ZCA_DATA = whiten(X, method='zca')
image_plotting(ZCA_DATA[0:15].reshape(15,28,28))

"""# CNN"""

"""## Defs"""
from tensorflow import keras
from keras.utils import to_categorical

train_labels = to_categorical(train_labels)
test_labels = to_categorical(test_labels)

from keras.models import Sequential
from keras.layers import Dense, Dropout, Activation, Flatten
from keras.layers import Conv2D, MaxPooling2D

""" The first convolutional neural network architecture"""
def create_new_model():
    model = Sequential()
    model.add(Conv2D(64, (3, 3), input_shape=(28, 28, 1) ))
    model.add(Activation('relu'))
    model.add(MaxPooling2D(pool_size=(2, 2)))

    model.add(Conv2D(64, (3, 3)))
    model.add(Activation('relu'))
    model.add(MaxPooling2D(pool_size=(2, 2)))

    model.add(Flatten())
    model.add(Dense(64))

    model.add(Dense(10))
    model.add(Activation('softmax'))
```

```
84
85      model.compile(loss=keras.losses.categorical_crossentropy,
86                    optimizer=keras.optimizers.Adam(),
87                    metrics=['accuracy'])
88      print(model.summary())
89      return model
90
91  """ The second convolutional neural network architecture
92  def create_new_model():
93      model = Sequential()
94      model.add(Conv2D(32, (3, 3), input_shape=(28, 28, 1) ))
95      model.add(Activation('relu'))
96      model.add(MaxPooling2D(pool_size=(2, 2)))
97
98      model.add(Flatten())
99      model.add(Dense(32))
100
101     model.add(Dense(10))
102     model.add(Activation('softmax'))
103
104     model.compile(loss=keras.losses.categorical_crossentropy,
105                   optimizer=keras.optimizers.Adam(),
106                   metrics=['accuracy'])
107     print(model.summary())
108     return model
109     """
110
111 def train(model, train_data, epochs):
112     history = model.fit(train_data, train_labels, epochs=epochs)
113     acc = history.history['accuracy']
114     loss = history.history['loss']
115     epochs = range(len(acc))
116     return  acc
117
118 def testing(model, test_data):
119     test_loss, test_acc = model.evaluate(test_data,  test_labels)
120     print('Test loss', test_loss)
121     print('Test accuracy', test_acc)
122     pass
123
124 def prediction(model, test_data):
125     predictions = model.predict(test_data)
126     mean = np.argmax(np.round(predictions[0]))
```

```
    plt.imshow(test_data[0].reshape(28, 28), cmap = plt.cm.binary)
    plt.show()
    return print('Model prediction: '+ str(mean))

"""## CNN with initial data"""

train_img, test_img = np.split(X, [60000])
train_img = train_images.reshape(60000, 28, 28, 1)
test_img = test_images.reshape(10000, 28, 28, 1)

first_model = create_new_model()
acc_init = train(first_model, train_img, 10)
testing(first_model, test_img)

"""## CNN with PCA-whitened data"""

PCA_train_img, PCA_test_img = np.split(PCA_DATA, [60000])
PCA_train_img = PCA_train_img.reshape(60000, 28, 28, 1)
PCA_test_img = PCA_test_img.reshape(10000, 28, 28, 1)

PCA_model = create_new_model()
acc_pca = train(PCA_model, PCA_train_img, 10)
testing(PCA_model, PCA_test_img)
prediction(PCA_model, PCA_test_img)

"""## CNN with ZCA-whitened data"""

ZCA_train_img, ZCA_test_img = np.split(ZCA_DATA, [60000])
ZCA_train_img = ZCA_train_img.reshape(60000, 28, 28, 1)
ZCA_test_img = ZCA_test_img.reshape(10000, 28, 28, 1)

ZCA_model = create_new_model()
acc_zca = train(ZCA_model, ZCA_train_img, 10)
testing(ZCA_model, ZCA_test_img)
prediction(ZCA_model, ZCA_test_img)

"""## CNN with randomly converted data"""

RAND_DATA = whiten(X, method='random')

train_img_rand, test_img_rand = np.split(RAND_DATA, [60000])
train_img_rand = train_img_rand.reshape(60000, 28, 28, 1)
test_img_rand = test_img_rand.reshape(10000, 28, 28, 1)
```

```
170
171 fourth_model = create_new_model()
172 acc_rand = train(fourth_model, train_img_rand, 10)
173 testing(fourth_model, test_img_rand)
174 prediction(fourth_model, test_img_rand)
175
176 """# Graph of accuracy during training"""
177
178 epochs = range(len(acc_init))
179     plt.plot(epochs, acc_init,  label="Init data") #blue
180     plt.plot(epochs, acc_pca,  label='PCA') #orange
181     plt.plot(epochs, acc_zca,  label='ZCA') #green
182     plt.plot(epochs, acc_rand,  label='Random') #red
183     plt.legend()
184     plt.xlabel('Epoch')
185     plt.ylabel('Accuracy')
186     plt.title('Training accuracy')
187     plt.grid(True)
188     #plt.figure()
189     plt.savefig("Training accuracy")
```