ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ОБРАЗОВАНИЯ
«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ
”ВЫСШАЯ ШКОЛА ЭКОНОМИКИ”»

Московский институт электроники и математики

Юткин Дмитрий Игоревич, группа БИВ-141

ПРИМЕНЕНИЕ ГЛУБОКИХ СВЁРТОЧНЫХ НЕЙРОННЫХ СЕТЕЙ ДЛЯ РЕШЕНИЯ ЗАДАЧИ РАСПОЗНАВАНИЯ ОБРАЗОВ

Междисциплинарная курсовая работа
по направлению 09.03.01.62 Информатика и вычислительная техника
студента образовательной программы бакалавриата
«Информатика и вычислительная техника»

Студент _____ Д.И. Юткин

Научный руководитель
Старший преподаватель
Д.В. Пантюхин

_____

Москва 2016 г.

Аннотация

Работа посвящена решению задачи распознавания образов с использованием глубоких свёрточных нейронных сетей. В результате исследования было обучено несколько нейронных сетей различной глубины (от 9 до 18 слоёв), часть из которых обучалась на предварительно обработанных данных. Максимальная точность одиночной модели составила 93,4%, объединение нейронных сетей в ансамбль позволило увеличилась точность распознавания до 94,21%, что сравнимо с точностью человека. Все эксперименты проводились на наборе изображений CIFAR-10, с использованием графических процессоров и фреймворка для глубокого обучения Caffe.

Abstract

The research focused on application of deep convolutional neural networks for solving object recognition task in natural images. As a result, several models of different depth (from 9 to 18 layers) have been trained on preprocessed and augmented data. The best model has shown accuracy of 93,4% on a test images. Whereas ensemble of combined models has achived 94,21% which is close-to-human performance. All experiments have been carried out on the CIFAR-10 dataset with modern GPUs and Caffe framework for a deep learning.

Основные термины, определения и сокращения

API (Application Programming Interface) — набор функций, процедур, структур и констант, предоставляемых сервисом или библиотекой стороннему разработчику.

BLAS (Basic Linear Algebra Subprograms) — API для создания приложений, выполняющих базовые операции линейной алгебры.

Batch normalization — метод, используемый для нормализации данных «протекающих» в нейронной сети в процессе её обучения.

CLI (Command Line Interface) — тип интерфейса, в котором инструкции передаются компьютеру путём ввода с клавиатуры.

CHH (Свёрточные нейронные сети) — специализированный тип нейронных сетей, который служит для работы с данными, имеющими пространственную топологию. Например, временные ряды или изображения. Нейросеть называется свёрточной, если использует хотя бы в одном из своих слоёв математическую операцию свёртки.

cuBLAS — реализация BLAS для NVIDIA GPUs.

cuDNN — библиотека для работы с примитивами глубоких нейронных сетей (свёртка, pooling, нормализация, активации и тд.) на GPUs.

L2 regularization — один из способов регуляризации. Реализуется путём добавления слагаемого $\frac{1}{2}\lambda W^2$ в шаге SGD, где λ — коэффициент регуляризации, W — параметры модели.

LMDB (Lightning Memory-Mapped Database) — программное обеспечение, предоставляющее высокопроизводительные базы данных, с упорядоченным хранилищем вида ключ-значение.

ReLu (Rectified Linear Unit) — функция активации вида $f(x) = \max(0, x)$.

Leaky ReLU — модификация ReLU. Имеет вид $f(x) = \alpha x$, если $x < 0$ и $f(x) = x$, если $x \geq 0$.

Maxout unit — функция активации вида $h_i(x) = \max_{j \in [1,k]}(x^T W_{ij} + b_{ij})$. Данная функция вычисляет максимум из k своих входов.

Mini batch — небольшая группа входных примеров (в свёрточных нейронных сетях — изображения), которая используется при вычислении ошибки в одной итерации SGD.

LR (learning rate) — задаёт размер шага в алгоритме градиентного спуска. Влияет на скорость обучения нейронной сети.

SGD (Stochastic gradient descent with momentum) — стохастическая аппроксимация градиентного спуска, в которой ошибка вычисляется не на всём обучающем множестве, а на каком то случайном его подмножестве, называемое mini batch. Шаг обновления весов выглядит следующим образом:

$$v_{t+1} = \mu v_t - \gamma \nabla E(z_i, w_t),$$
$$w_{t+1} = w_t + v_t,$$

где w_i — веса модели на i-ой итерации, γ — learning rate, E — функционал ошибки, z_i — mini batch, t — номер итерации, v_i — момент.

NAG (Nesterov accelerated gradient) — модификация алгоритма стохастического градиентного спуска. В отличии от SGD, в NAG формула для обновления весов имеет следующий вид:

$$v_{t+1} = \mu v_t - \gamma \nabla E(z_i, w_t + \mu v_t),$$
$$w_{t+1} = w_t + v_t,$$

где w_i — веса модели на i-ой итерации, γ — learning rate, E — функционал ошибки, z_i — mini batch, t — номер итерации, v_i — момент.

Pooling — один из слоёв свёрточных нейронных сетей. Выполняет уменьшение размерностей активаций предыдущего свёрточного слоя.

Stacking — один из способов построения ансамбля. Идея «стэкинга» заключается в обучении алгоритма, делающего финальные предсказания, на ответах базовых моделей.

Stride — один из параметров свёрточного слоя. Отвечает за размер шага, с которым ядро свёртки «перемещается» вдоль входного изображения.

ZCA Whitening (Zero-phase Component Analysis Whitening) — алгоритм препроцессинга данных. Если входные данные являются изображениями, то данный метод избавляет их от лишней информации, уменьшает взаимную корреляцию и добивается их одинаковой дисперсии.

Оглавление

1 Введение

В последние годы компьютерное зрение является одной из самых активно развивающихся областей искусственного интеллекта. Подобный интерес к области обусловлен недавними успехами в обучении компьютера воспроизводить зрительные способности человека, например, распознавать и классифицировать образы.

До недавнего времени распознавание объектов осуществлялось с помощью алгоритмов, для которых вручную приходилось проектировать признаки (feature engineering). Основной недостаток такого подхода — невозможность находить в изображении средние и многоуровневые абстракции, такие как части объекта или пересечения различных краёв. Кроме того, признаки созданные вручную для одного типа изображений зачастую не были применимы к другим. Однако, недавние разработки в области машинного обучения, известные как глубокое обучение (deep learning) [1], показали, как вычислительные модели, состоящие из множества слоёв, могут автоматически изучать иерархии признаков из набора данных.

На сегодняшний день глубокое обучение развилось и укрепилось в отдельную ветвь машинного обучения, алгоритмы которой способствовали значительному улучшению результатов в различных задачах, таких как обработка естественного языка, распознавание визуальных образов и др.

Свёрточные нейронные сети (СНН) — одна из главных причин прорыва в компьютерном зрении. Изначально представленные в 1980 году Кунихикой Фукусимой как NeoCognitron [2], а затем улучшенные в 1998 году Яном Лекуном до LeNet-5 [3], СНН получили славу благодаря впечатляющему успеху в распознавании рукописных цифр. Для следующего прорыва в компьютерном зрении потребовалось чуть больше двадцати лет. С увеличением производительности графических процессоров (GPU), стало возможным обучение глубоких нейронных сетей. В 2012 году глубокая СНН победила в мировом соревновании ImageNet Large-scale Visual Recognition Challenge (ILSVRC), значительно превзойдя предыдущие результаты, достигнутые алгоритмами полагающимися на ручную генерацию признаков [4].

Таким образом, темой данного исследования является разработка системы распознавания объектов (object recognition), основанной на глубоких свёрточных нейронных сетях. Основные цели исследования:

а) Изучение и применение на практике свёрточных нейронных сетей в решении задачи распознавания объектов;

б) Изучение современных технологий машинного обучения, анализа данных и глубокого обучения.

Для достижения поставленных целей были сформулированы следующие задачи:

а) Проведение обзора статей и литературы, посвящённых свёрточным нейронным сетям и решению задачи распознавания образов;
б) Выбор и изучение программного обеспечения для глубокого обучения;
в) Выбор набора данных, на котором будут проводиться эксперименты;
г) Проектирование и обучение нейронных сетей различных архитектур;
д) Оценка и сравнение точностей полученных нейронных сетей;
е) Объединение отдельно обученных моделей в ансамбль.

Все поставленные задачи выполнены, цели исследования достигнуты. Результаты каждой из задач будут описаны далее, в основной части работы.

2 Основная часть

2.1 Оборудование, использовавшиеся в работе

Традиционно, нейронные сети обучались с использованием CPU на одной машине. Сегодня такой подход является неэффективным. Современные библиотеки для работы с глубокими нейронными сетями используют GPU или распределённые CPU вычисления между несколькими машинами. Свёрточные сети зачастую хранят огромное число буферов параметров, активаций и градиентов. Все эти значения должны быть обработаны в течении каждого шага обучения. Количество таких буферов может быть настолько велико, что может не помещаться в кэш традиционных десктопных компьютеров. Напротив, пропускная способность памяти графических процессоров намного выше, чем в CPU. Кроме того, алгоритмы нейронных сетей зачастую не требуют сложных ветвлений или рекурсивных вызовов, что позволяет легко исполнять их на GPU. В силу того, что нейронные сети могут быть разделены на множество индивидуальных нейронов, которые способны проводить вычисления независимо от других нейронов в слое, нейронные сети получают большую выгоду от использования GPU параллелизма. [5, стр. 448] Таким образом, использование специализированного оборудования является важным фактором успешного исследования.

В данной работе эксперименты выполнялись с использованием веб-сервиса Amazon Elastic Compute Cloud, который «предоставляет масштабируемые вычислительные ресурсы в облаке и упрощает процесс крупномасштабных вычислений для разработчиков»[1]. Сервер, на котором обучались нейронные сети (g2.2xlarge), имеет следующие характеристики:

- CPU Intel Xeon E5-2670 2.6 GHz, 8 ядер;
- GPU NVIDIA GRID K520 4 Gb, 1536 CUDA ядер;
- 16 Gb RAM.

2.2 Датасет CIFAR-10

Набор данных, на котором проводились исследования, состоит из 60 000 цветных изображений, размера 32×32 пикселя. Каждое изображение принадлежит одному из 10 классов[2], что соответствует 6 000 изображениям на класс. Под обуче-

[1] https://aws.amazon.com/ec2/

[2] самолёт, автомобиль, птица, кот, олень, собака, лягушка, лошадь, корабль, грузовик

ние отводится 50 000 изображений. Остальные 10 000 используются для тестирования. Объекты в классах сильно варьируются, например, класс «птица» содержит различные виды птиц, как большие так и маленькие. Кроме того, объекты классов представлены в различных позах и под различными углами. Особенно это проявляется среди собак и котов, которые изображены не только в различных позах, но иногда и частично, например, изображена только голова животного.

Датасет CIFAR-10 [6] был выбран для исследований благодаря своему относительно небольшому размеру, который позволил обучать глубокие свёрточные нейронные сети, быстро проводить эксперименты и проверять на практике различные гипотезы.

На момент написания работы лучший результат (state-of-the-art) на CIFAR-10 95,53% [7]. Точность распознавания человека $\approx 94\%$.[3]

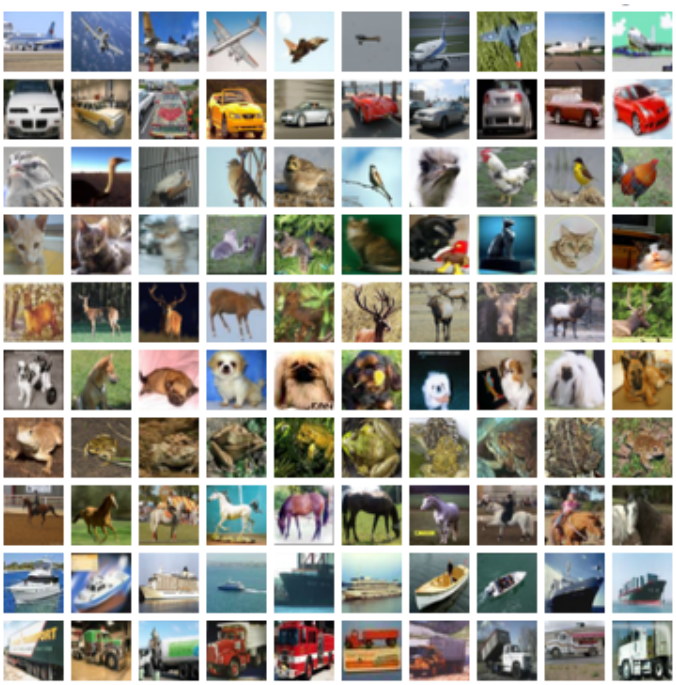

Рисунок 1 — 10 случайных изображений из каждого класса

2.3 Фреймворк Caffe

Свёрточные нейронные сети обучались с помощью фреймворка для глубокого обучения Caffe [8]. Изначально Caffe разрабатывался командой BVLC (Berkeley Vision and Learning Center), но постепенно перерос в большой open-source проект.[4]

[3] http://karpathy.github.io/2011/04/27/manually-classifying-cifar10/
[4] https://github.com/BVLC/caffe

На данный момент вклад в развитие Caffe внесли почти 200 разработчиков и более 10 000 человек оценили проект на Github.

Данный фреймворк был выбран для решения задачи по нескольким причинам:

а) Простота определения моделей и методов оптимизации. Архитектуры нейронных сетей (Caffe поддерживает топологии сетей в форме любых ациклических графов) и алгоритмы оптимизации удобно и эффективно определяются в специальных конфигурационных файлах типа Google Protocol Buffers[5].

б) Модульность. Caffe позволяет легко изменять архитектуру сети под новые форматы входных данных. Кроме того, в наличии имеется много слоёв и функций потерь.

в) Скорость вычислений и эффективное использование ресурсов. Caffe заранее выделяет ровно столько памяти сколько нужно для нейронной сети. Операций линейной алгебры такие как умножение, сложение и свёртка выполняются на CPU с помощью BLAS (Basic Linear Algebra Subroutines). На GPU за эти операции отвечают библиотеки cuBLAS и cuDNN [9]

г) CLI и интерфейсы для Python и Matlab.

Caffe написан на языках C++, Cuda, Python. Для хранения датасетов используются эффективные базы данных LMDB[6] и LevelDB[7]. Обучение нейронных сетей может выполняться как на CPU, так и на нескольких GPU одновременно. Фреймворк доступен для Linux, Windows и OS X.

2.4 Предварительная обработка данных

Необработанные изображения содержат излишнюю информацию, так как смежные пиксели имеют высокую корреляцию. Поэтому прежде чем подавать изображения на вход нейронной сети, они были обработаны в два этапа. В начале была произведена глобальная нормализация контраста (global contrast normalization):

$$\widehat{X} = \frac{X - \overline{X}}{\sigma},$$

[5] https://developers.google.com/protocol-buffers/

[6] http://symas.com/mdb/

[7] https://github.com/google/leveldb

где X — исходное изображение, $\overline{X}$ — среднее значение, σ — стандартное отклонение.

Затем изображения были линейно трансформированы с помощью ZCA whitening [6]. Цель данного алгоритма сделать входные изображения слабо коррелирующими друг с другом.

Листинг 1 — Алгоритм ZCA whitening

```
cov = np.dot(X.T, X) / X.shape[0] # Вычисляем ковариационную матрицу
U,S,V = np.linalg.svd(cov) # Находим сигнулярное разложение ковариацонной матрицы
Xrot = np.dot(X, U) # Поворачиваем входные данные
Xwhite = Xrot / np.sqrt(S + α) # Делим на собственные числа
```

Рисунок 2 — Изображения после применения ZCA whitening с параметром $\alpha = 0{,}1$

Предобработка с помощью ZCA whitening уменьшила ошибку в среднем на $1{,}5\%$. Кроме того, каждое изображение было увеличено до 40×40 пикселей, для того чтобы в процессе обучения вырезать из изображения случайный патч размером 32×32. Такой приём позволяет снизить эффект переобучения модели и повысить конечную точность распознавания.

2.5 Обучение нейронных сетей

Во время исследования были проверены различные гипотезы, архитектуры и эвристики так или иначе влияющие на конечные точности моделей. В результате различных экспериментов были отобраны семь нейронных сетей, показавшие

наилучший результат. В приложении отражены топологии сетей (таблица 1), графики обучения (рисунок 17) и время, затраченное на обучение каждой из моделей (рисунок 18).

2.5.1 Инициализация параметров

Оптимизационные алгоритмы для глубоких нейронных сетей являются итеративными и, таким образом, требуют от пользователя задания начальной точки, с которой нужно начать обучение. От выбора этой точки определятся конечная сходимость алгоритма. Более того, выбор начальной точки определяет с какой скоростью будет проходить обучение и какова будет ошибка в конечной точке, к которой сойдётся оптимизационный алгоритм.

Современные стратегии инициализации просты и основаны на различных эвристиках. Проектирование более сложных методов является трудной задачей, так как оптимизация нейронных сетей ещё не достаточно изучена. В данной работе инициализация проводилась с помощью layer-sequential unit-variance [10]. Данный метод сначала проводит преинициализацию случайными ортонормированными матрицами [11], затем, последовательно для каждого слоя настраивает весовые коэффициенты таким образом, чтобы дисперсия выходных данных слоя была единичной. Эксперименты показали, что такая инициализация позволяет эффективно обучать очень глубокие нейронные сети. Далее будут описаны отобранные для конечного ансамбля модели, их архитектуры, свойства и особенности обучения.

2.5.2 Модель I

Модель I состоит из 18 обучающихся слоёв и $\approx 2{,}7$ млн. параметров. Данная нейронная сеть достигла самой высокой точности предсказания среди всех обученных моделей (93,4%). Maxout в качестве функции активации является её отличительной особенностью. На рисунке 3 показано как maxout соединён с выходами свёрточных слоёв. На рисунке 4 можно видеть соединение выходов полносвязного слоя и активации.

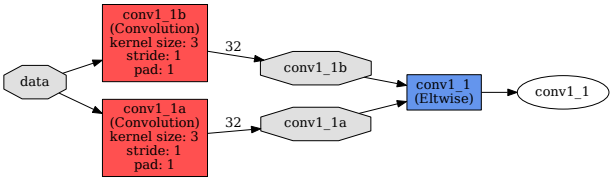

Рисунок 3 — Соединение maxout со свёрточными слоями

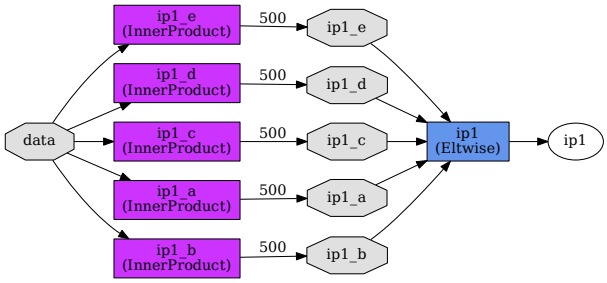

Рисунок 4 — Соединение maxout с полносвязными слоями

Нейронная сеть обучалась 90 000 итераций методом стохастического градиентного спуска, с начальным learning rate (lr) 0,01, моментом 0.9, L2 регуляризацией с коэффициентом 0,0005 и мини батчем (mini batch) из 256 изображений. В процессе обучения, начиная с 40 000 итерации, lr уменьшался в десять раз каждые 20 000 итераций, достигнув к концу обучения значения 10^{-5}.

	Plane	Car	Bird	Cat	Deer	Dog	Frog	Horse	Ship	Truck
Plane	943	4	13	9	1	0	2	1	16	11
Car	3	967	1	0	0	1	0	0	6	22
Bird	18	0	910	16	15	18	13	2	6	2
Cat	13	2	15	856	13	64	20	11	2	4
Deer	4	0	16	15	933	16	4	10	1	1
Dog	2	1	9	60	14	903	3	5	0	3
Frog	4	1	13	7	8	7	958	0	1	1
Horse	2	0	4	12	17	11	3	950	1	0
Ship	19	5	3	2	0	1	5	0	960	5
Truck	6	23	2	1	0	0	0	0	8	960

Рисунок 5 — Матрица предсказаний модели I

2.5.3 Модели II и III

Модель II имеет в 4,5 раза меньше параметров, в сравнении с моделью I ($\approx 600\,000$). Данная нейронная сеть имеет классическую архитектуру — чередова-

ние свёрточных и pooling слоёв, в качестве функции активации используется Leaky ReLU[8] с параметром $\alpha = 0{,}01$.

Сеть обучалась в течении 30 000 итераций. Функционал ошибки оптимизировался алгоритмом NAG (Nesterov accelerated gradient) [12], с начальным learning rate (lr) 0,01, моментом 0.9, L2 регуляризацией с коэффициентом 0,0005 и мини батчем из 256 изображений. Начиная с 15 000 итерации, lr уменьшался в десять раз каждые 5 000 итераций, достигнув к концу обучения значения 10^{-5}.

Результат модели II оказался ниже на тестовом множестве, в сравнении с моделью I (89,48% против 93,4%). Данный результат объясняется недообучением, что является следствием недостаточного числа параметров в нейронной сети. С другой стороны, за счёт уменьшения числа весов в модели, удалось снизить время обучения в 10 раз (с десяти часов до одного).

Модель III является попыткой исправить главный недостаток модели II за счёт увеличения числа параметров. В связи с чем, число фильтров каждого свёрточного слоя было увеличено в два раза, что позволило снизить ошибку на 1,55%. Параметры оптимизации использовались такие же как в модели II.

	Plane	Car	Bird	Cat	Deer	Dog	Frog	Horse	Ship	Truck
Plane	894	10	22	6	5	1	1	4	42	15
Car	3	956	2	0	1	1	1	0	8	28
Bird	21	1	856	27	36	20	22	10	5	2
Cat	10	3	36	738	38	109	29	20	10	7
Deer	4	0	24	14	910	12	13	20	1	2
Dog	7	3	25	78	20	840	7	17	0	3
Frog	6	2	16	19	12	4	937	1	1	2
Horse	5	1	8	6	25	26	1	923	2	3
Ship	21	7	3	3	2	0	4	1	948	11
Truck	8	33	1	2	0	0	2	0	8	946

Рисунок 6 — Матрица предсказаний модели II

	Plane	Car	Bird	Cat	Deer	Dog	Frog	Horse	Ship	Truck
Plane	913	4	21	6	6	1	2	3	31	13
Car	5	970	0	0	1	0	2	0	3	19
Bird	21	0	876	18	30	23	19	7	3	3
Cat	6	2	30	809	25	74	31	12	5	6
Deer	2	1	19	17	914	12	13	18	1	3
Dog	3	0	16	90	18	852	10	9	1	1
Frog	5	2	17	22	8	2	942	0	1	1
Horse	4	0	10	10	22	22	3	926	1	2
Ship	22	7	3	3	2	0	1	0	952	10
Truck	8	30	1	0	1	0	1	0	10	949

Рисунок 7 — Матрица предсказаний модели III

[8] $f(x) = \max(x, \alpha x)$

2.5.4 Модели IV и V

На моделях IV и V проверялась гипотеза, что pooling слои могут быть заменены на соответствующие свёрточные слои с увеличенным параметром stride без значительного снижения точности распознавания. [13]. В данных сетях max-pooling 2×2 слои были заменены на свёрточные слои со stride равным двум.

Модель IV имеет ≈6 млн. параметров и 13 слоёв. В качестве функции активации используется ReLU. Параметры оптимизации использовались такие же как и в моделях II и III, за исключением увеличенного до 50 0000 числа итераций SGD. После обучения в течении 14 часов, точность на тестовом множестве достигла 92,1%, что подтверждает гипотезу о замене pooling слоёв.

Модель V состоит из 16 слоёв и ≈7,8 млн свободных параметров. Данная нейронная сеть является попыткой улучшить точность модели IV за счёт увеличения числа свёрточных слоёв. Однако, дополнительные слои привнесли дополнительно ≈1,8 млн. параметров, что привело к переобучению и снижении конечной точности на 1,19%. Модель V обучалась в течении 19 часов, с мини батчем из 128 изображений и используя такие же оптимизационные параметры как и модель IV.

	Plane	Car	Bird	Cat	Deer	Dog	Frog	Horse	Ship	Truck
Plane	918	4	24	7	5	0	1	2	31	8
Car	5	960	0	2	1	1	1	0	6	24
Bird	17	0	888	20	23	19	15	9	5	4
Cat	9	1	24	824	24	76	18	15	6	3
Deer	2	0	9	17	935	16	5	10	4	2
Dog	0	0	14	63	15	891	2	14	0	1
Frog	7	1	17	22	4	1	942	1	4	1
Horse	5	0	7	4	17	16	1	947	1	2
Ship	21	6	2	4	1	0	2	0	957	7
Truck	8	25	1	3	0	1	3	0	10	949

Рисунок 8 — Матрица предсказаний модели IV

	Plane	Car	Bird	Cat	Deer	Dog	Frog	Horse	Ship	Truck
Plane	923	6	17	12	1	0	0	3	26	12
Car	3	967	1	2	0	2	0	0	4	21
Bird	17	0	885	31	19	26	8	11	3	0
Cat	6	1	19	845	15	80	13	12	5	4
Deer	4	0	26	30	889	17	7	24	2	1
Dog	3	2	19	97	10	850	2	17	0	0
Frog	8	2	26	44	10	11	896	0	2	1
Horse	6	0	10	14	7	14	1	948	0	0
Ship	25	5	3	7	0	1	1	0	950	8
Truck	7	35	1	6	0	0	2	2	8	939

Рисунок 9 — Матрица предсказаний модели V

2.5.5 Модель VI

Модель VI состоит из 18 слоёв и ≈4,2 млн. параметров. Данная модель использует слои batch normalization, которые преобразовывают активации свёрточ-

ных и полносвязных слоев таким образом, что их среднее равно нулю, а дисперсия единице (значения вычисляются на всём mini batch). Авторы batch normalization [14] утверждают, что данный приём позволяет использовать высокий learning rate и приводит к более быстрой сходимости. Данная нейронная сеть обучалась 70 000 итераций с помощью SGD. Модель VI достигла точности в 92,0% за 25 0000 итераций, обучение сети заняло 10 часов.

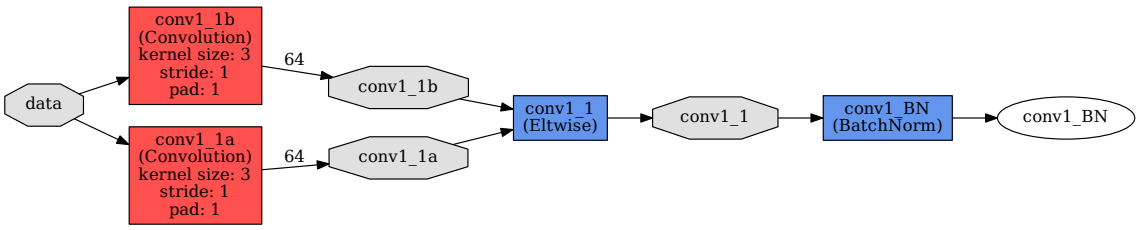

Рисунок 10 — Batch normalization слой в модели VI

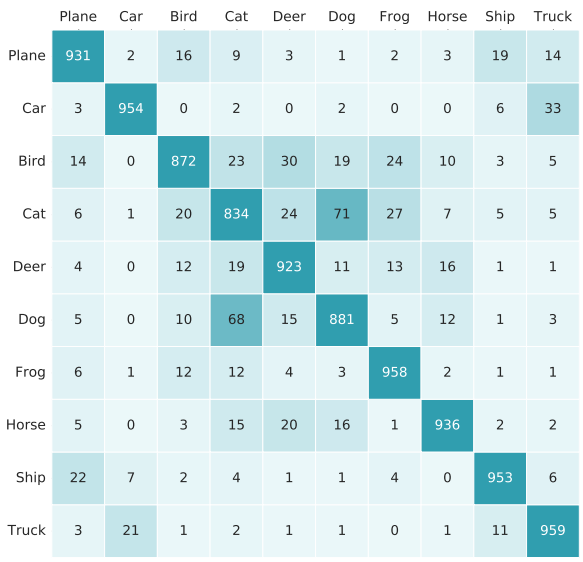

Рисунок 11 — Матрица предсказаний модели VI

2.5.6 Модель VII

Модель VII идентична модели I, за исключением количества свёрточных слоёв соединённых с maxout активацией. В данной модели их пять (Рис. 12), тогда как модель I имеет только два слоя. Такое изменение в топологии сети не привело к увеличению конечной точности модели, более того, ошибка увеличилась на 0,9%. Максимальное число итераций достигло 70 000, время затраченное на обучение составило 22 часа.

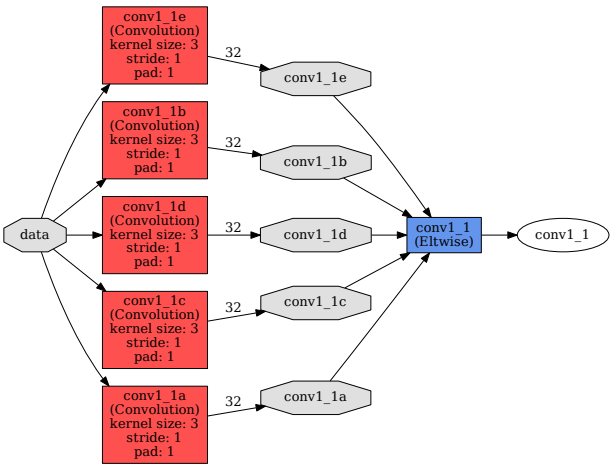

Рисунок 12 — Соединение maxout активаций в модели VII

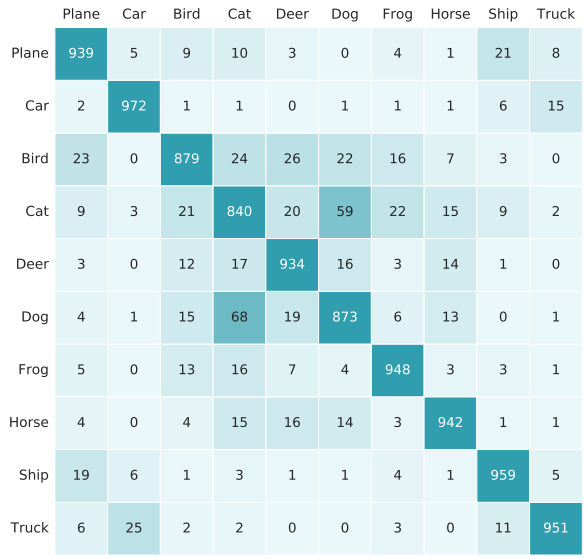

Рисунок 13 — Матрица предсказаний модели VII

2.6 Объединение нейронных сетей в ансамбль

Объединение нескольких моделей в ансамбль является мощным приёмом при решении различных задач машинного обучения. Применение подобной техники было выбрано по причине наличия различных свёрточных нейронных сетей, предсказания которых не очень сильно коррелируют между собой. В данном исследовании создание ансамбля производилось при помощи «стэкинга» (stacking) [15], идея которого заключается в построении алгоритма, объединяющего предсказания базовых моделей (в данном случае нейронных сетей). Такой подход зачастую работает лучше, так как усиливает обобщающую способность и робастность конечного классификатора за счёт одиночных моделей.

Для построения конечного ансамбля были протестированы различные алгоритмы классификации, такие как случайный лес, метод k ближайших соседей, машина опорных векторов и Extra Trees. Наилучшая точность была достигнута с помощью алгоритма Extra Trees, [16] (94,21%), который сам является ансамблем деревьев решений. Схема объединения моделей отражена на рисунке 14.

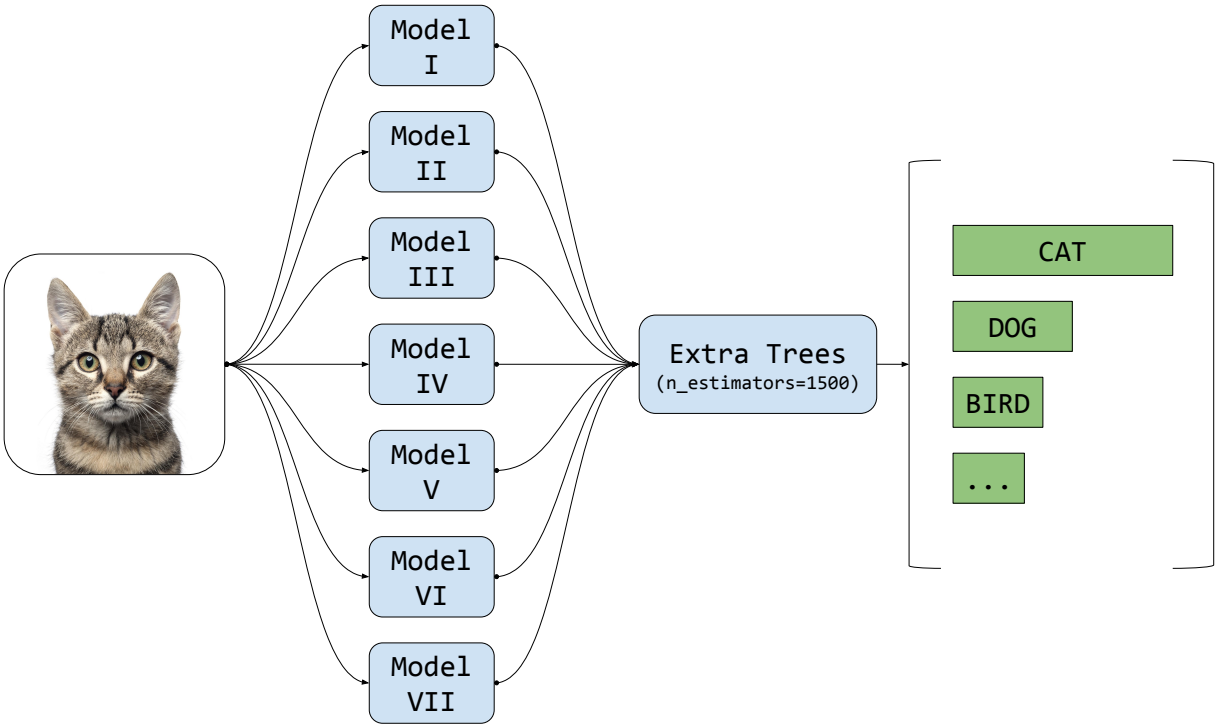

Рисунок 14 — Схема конечного ансамбля

	Plane	Car	Bird	Cat	Deer	Dog	Frog	Horse	Ship	Truck
Plane	956	3	6	9	1	0	2	1	14	8
Car	2	978	0	0	0	0	0	0	3	17
Bird	16	0	908	18	20	16	11	7	3	1
Cat	8	2	15	879	14	48	22	7	3	2
Deer	2	0	11	13	950	9	4	11	0	0
Dog	4	0	10	57	12	906	3	7	0	1
Frog	4	1	10	10	5	2	965	1	1	1
Horse	5	0	2	11	14	14	2	950	1	1
Ship	18	6	1	5	0	0	2	0	964	4
Truck	4	19	1	2	0	0	0	0	9	965

Рисунок 15 — Матрица предсказаний ансамбля моделей

3 Заключение

В результате выполнения курсовой работы был спроектирован и обучен ансамбль моделей, состоящий из семи различных глубоких свёрточных нейронных сетей и решающий задачу распознавания образов на датасете CIFAR-10 с точностью, превосходящей точность распознавания человека.

В процессе исследования были получены теоретические знания в области машинного обучения, анализа данных и глубокого обучения. В частности, были подробно изучены свёрточные нейронные сети, способы их проектирования, настройки и оптимизации, а также фреймворк Caffe, позволяющий реализовывать модели и современные алгоритмы компьютерного зрения. Полученные знания послужат фундаментом для дальнейших исследований в этой области. Таким образом, по результатам работы можно сказать, что все поставленные задачи выполнены, цели достигнуты.

Список литературы

1. *Sutskever Ilya*. A Brief Overview of Deep Learning. — 01.13.2015. http://goo.gl/lQVVmo.

2. *Fukushima Kunihiko*. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position // *Biological Cybernetics*. — Vol. 36, no. 4. — Pp. 193–202.

3. Gradient-Based Learning Applied to Document Recognition / Y. LeCun, L. Bottou, Y. Bengio, P. Haffner // *Proceedings of the IEEE*. — 1998. — November. — Vol. 86, no. 11. — Pp. 2278–2324.

4. *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. — Pp. 1097–1105.

5. *Ian Goodfellow Yoshua Bengio, Courville Aaron*. Deep Learning. — Book in preparation for MIT Press. http://www.deeplearningbook.org.

6. *Krizhevsky Alex*. Learning multiple layers of features from tiny images. — 2009.

7. *Graham B.* Fractional Max-Pooling // *ArXiv e-prints*. — 2014. — dec. http://adsabs.harvard.edu/abs/2014arXiv1412.6071G.

8. Caffe: Convolutional Architecture for Fast Feature Embedding / Yangqing Jia, Evan Shelhamer, Jeff Donahue et al. // *arXiv preprint arXiv:1408.5093*. — 2014.

9. cuDNN: Efficient Primitives for Deep Learning / Sharan Chetlur, Cliff Woolley, Philippe Vandermersch et al. // *CoRR*. — 2014. — Vol. abs/1410.0759. http://arxiv.org/abs/1410.0759.

10. *Mishkin Dmytro, Matas Jiri*. All you need is a good init // *CoRR*. — 2015. — Vol. abs/1511.06422. http://arxiv.org/abs/1511.06422.

11. *Saxe Andrew M., McClelland James L., Ganguli Surya*. Exact solutions to the nonlinear dynamics of learning in deep linear neural networks // *CoRR*. — 2013. — Vol. abs/1312.6120. http://arxiv.org/abs/1312.6120.

12. On the importance of initialization and momentum in deep learning / Ilya Sutskever, James Martens, George E. Dahl, Geoffrey E. Hinton. — 2013. — may. — Vol. 28, no. 3. — Pp. 1139–1147. http://jmlr.org/proceedings/papers/v28/sutskever13.pdf.

13. Striving for Simplicity: The All Convolutional Net / Jost Tobias Springenberg, Alexey Dosovitskiy, Thomas Brox, Martin A. Riedmiller // *CoRR*. — 2014. — Vol. abs/1412.6806. http://arxiv.org/abs/1412.6806.

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

15. *Wolpert David H.* Stacked Generalization // *Neural Networks*. — 1992. — Vol. 5. — Pp. 241–259.

16. *Geurts Pierre, Ernst Damien, Wehenkel Louis*. Extremely Randomized Trees // *Mach. Learn.* — 2006. — apr. — Vol. 63, no. 1. — Pp. 3–42. http://dx.doi.org/10.1007/s10994-006-6226-1.

Приложение

Model I	Model II	Model III	Model IV	Model V	Model VI	Model VII
conv1_1 3x3x32	conv1_1 3x3x32	conv1_1 3x3x64	conv1_1 3x3x96	conv1_1 3x3x96	conv1_1 3x3x64	conv1_1 3x3x32
conv1_2 3x3x32	conv1_2 3x3x32	conv1_2 3x3x64	conv1_2 3x3x96	conv1_2 3x3x96	conv1_2 3x3x64	conv1_2 3x3x32
conv1_3 3x3x32	conv1_3 3x3x32	conv1_3 3x3x64	conv1_3 3x3x96	conv1_3 3x3x96	conv1_3 3x3x64	conv1_3 3x3x32
conv1_4 3x3x48	pool 2x2	pool 2x2	conv1_4 3x3x96 stride 2	conv1_4 3x3x96	conv1_4 3x3x64	conv1_4 3x3x48
conv1_5 3x3x48	conv2_1 3x3x64	conv2_1 3x3x128	conv2_1 3x3x192	conv1_5 3x3x96 stride 2	conv1_5 3x3x64	conv1_5 3x3x48
pool 2x2	conv2_2 3x3x64	conv2_2 3x3x128	conv2_2 3x3x192	conv2_1 3x3x192	pool 2x2	pool 2x2
conv2_1 3x3x80	conv2_3 3x3x64	conv2_3 3x3x128	conv2_3 3x3x192	conv2_2 3x3x192	conv2_1 3x3x128	conv2_1 3x3x80
conv2_2 3x3x80	conv2_4 3x3x64	conv2_4 3x3x128	conv2_4 3x3x192 stride 2	conv2_3 3x3x192	conv2_2 3x3x128	conv2_2 3x3x80
conv2_3 3x3x80	pool 2x2	pool 2x2	conv3_1 3x3x384	conv2_4 3x3x192	conv2_3 3x3x128	conv2_3 3x3x80
conv2_4 3x3x80	conv3_1 3x3x128	conv3_1 3x3x256	conv3_2 3x3x384	conv2_5 3x3x192 stride 2	conv2_4 3x3x128	conv2_4 3x3x80
conv2_5 3x3x80	conv3_2 3x3x128	conv3_2 3x3x256	conv3_3 3x3x384	conv3_1 3x3x384	conv2_5 3x3x128	conv2_5 3x3x80
conv2_6 3x3x80	conv3_3 3x3x128	conv3_3 3x3x256	conv3_4 3x3x384 stride 2	conv3_2 3x3x384	conv2_6 3x3x128	conv2_6 3x3x80
pool 2x2	conv3_4 3x3x128	conv3_4 3x3x256	pool global max	conv3_3 3x3x384	pool 2x2	pool 2x2
conv3_1 3x3x128	pool global	pool 2x2	fc 500	conv3_4 3x3x384	conv3_1 3x3x256	conv3_1 3x3x128
conv3_2 3x3x128	fc 256	fc 256	softmax 10	conv3_5 3x3x384 stride 2	conv3_2 3x3x256	conv3_2 3x3x128
conv3_3 3x3x128	softmax 10	softmax 10		pool global max	conv3_3 3x3x256	conv3_3 3x3x128
conv3_4 3x3x128				fc 500	conv3_4 3x3x256	conv3_4 3x3x128
conv3_5 3x3x128				softmax 10	conv3_5 3x3x256	conv3_5 3x3x128
conv3_6 3x3x128					conv3_6 3x3x256	conv3_6 3x3x128
pool global max					pool global AVE	pool global AVE
fc 500					fc 512	fc 512
softmax 10					softmax 10	softmax 10

Таблица 1: Архитектуры обученных моделей

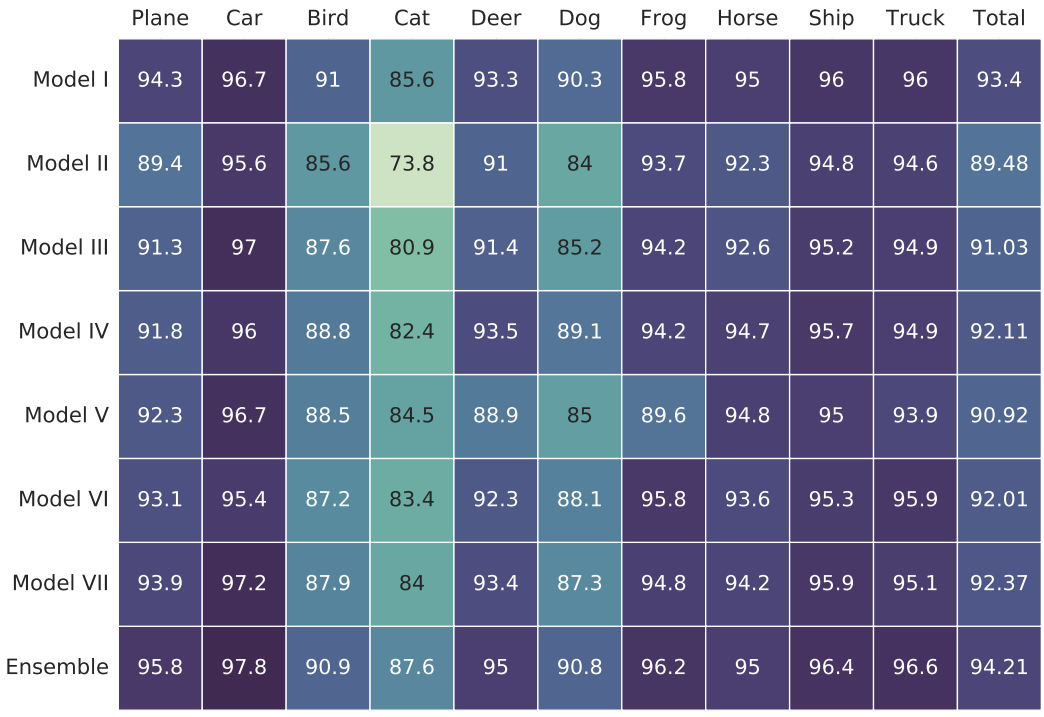

Рисунок 16 — Точности распознавания по различным классам

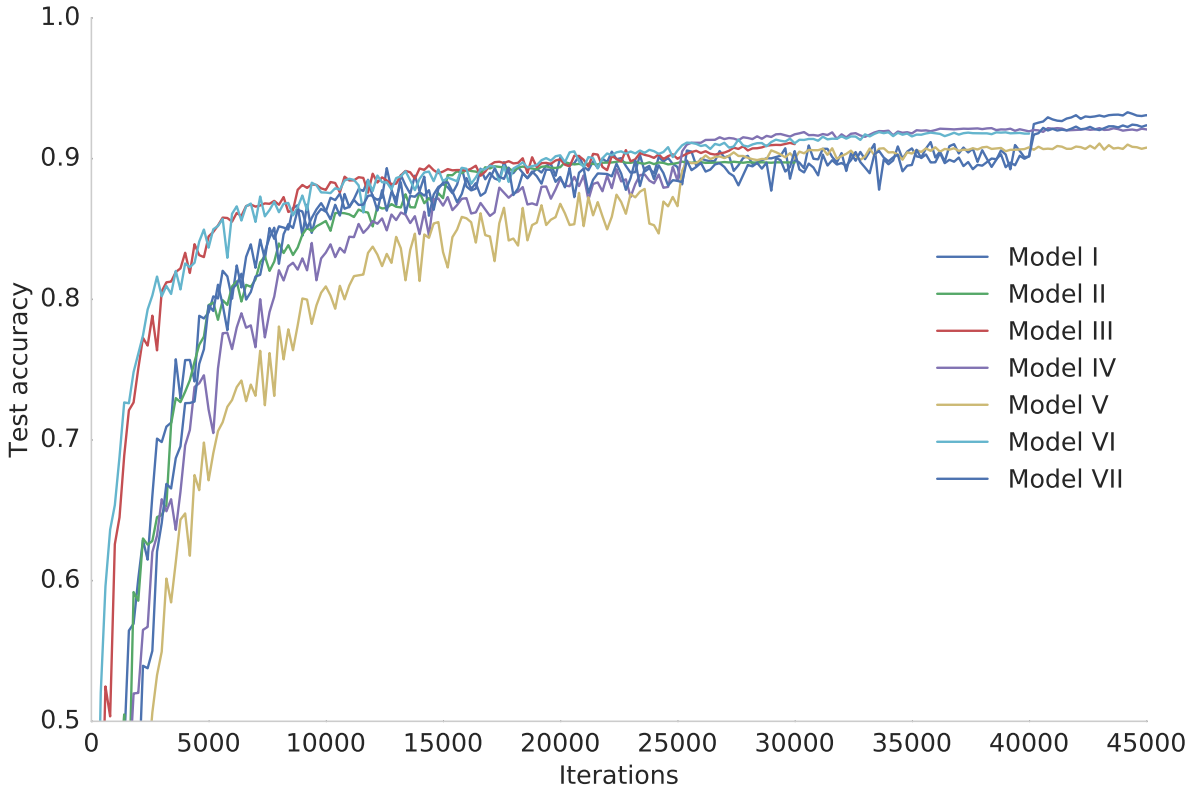

Рисунок 17 — Графики обучения моделей

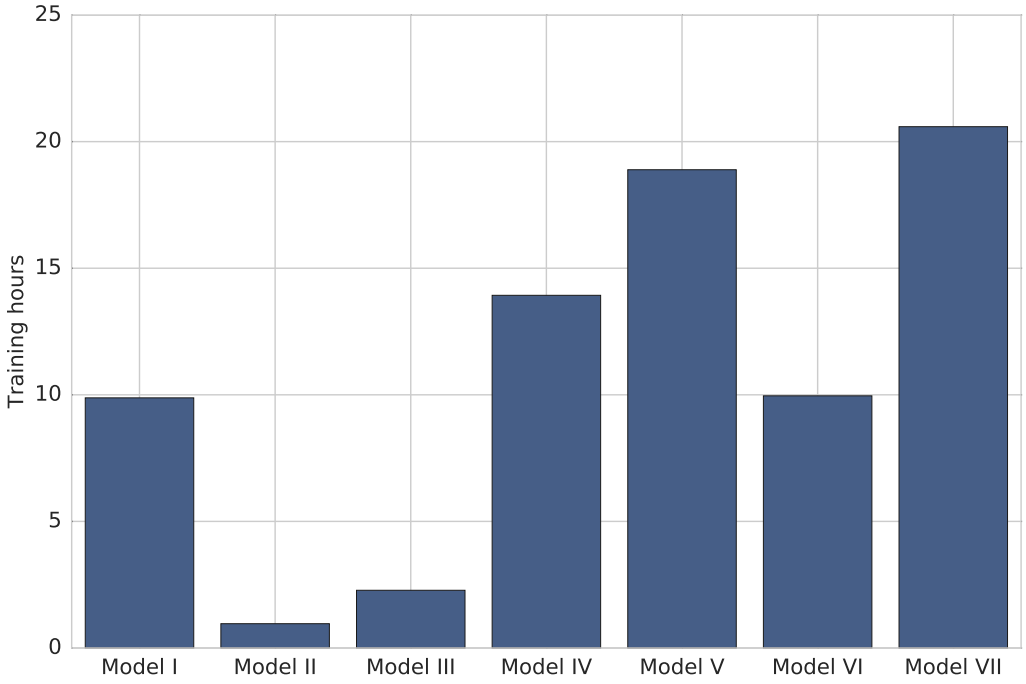

Рисунок 18 — Время, затраченное моделями на обучение

Листинг 2 — Препроцессинг изображений

```
from __future__ import print_function
import numpy as np
import scipy as sp
import os
import cPickle

class CIFAR10(object):
        def __init__(self, path):
        self.path = path
        f_meta = open(os.path.join(self.path, 'batches.meta'), 'r')
        self.meta = cPickle.load(f_meta)
        f_meta.close()
    def get_train_data(self):
        data, labels, fnames = [], [], []
        for i in xrange(1,6):
            f_batch = open(os.path.join(self.path, 'data_batch_%d' % i), 'r')
            batch = cPickle.load(f_batch)
            f_batch.close()
            for j in range(len(batch['filenames'])):
                data.append(np.array(batch['data'][j],
                                     dtype=float).reshape((3, 32, 32)))
                labels.append(batch['labels'][j])
                fnames.append(batch['filenames'][j])
        return np.array(data), np.array(labels), np.array(fnames)

    def get_test_data(self):
        data, labels, fnames = [], [], []
        f_batch = open(os.path.join(self.path, 'test_batch'), 'r')
        batch = cPickle.load(f_batch)
        f_batch.close()
        for i in range(len(batch['filenames'])):
            data.append(np.array(batch['data'][i],
                                 dtype=float).reshape((3, 32, 32)))
            labels.append(batch['labels'][i])
            fnames.append(batch['filenames'][i])
        return np.array(data), np.array(labels), np.array(fnames)

    def get_whitened_data(self, alpha, kind='train'):
        if kind == 'train':
            data, labels, fnames = self.get_train_data()
        elif kind == 'test':
            data, labels, fnames = self.get_test_data()
        else:
            raise RuntimeError('"%s" is not acceptible. '
                               'Either "test" or "train"' % kind)
```

```
        # ZCA
        N = data.shape[0]
        X = data.reshape((N,-1)) / 255
        X -= np.mean(X, axis=0)
        sigma = np.dot(X.T, X) / N
        U, S, V = np.linalg.svd(sigma)
        sqS = np.sqrt(S + alpha)
        Uzca = np.dot(U / sqS[np.newaxis, :], U.T)
        Z = np.dot(X, Uzca.T)
        zca_data = []
        for i in xrange(N):
            absmax = np.max(np.abs(Z[i, :]))
            tmp = Z[i, :].reshape((3, 32, 32)) / absmax*127 + 128
            zca_data.append(np.array(tmp.transpose(1,2,0), dtype=np.uint8))
        return np.array(zca_data), labels, fnames
```

Листинг 3 — Оценка модели и запись ответов в таблицу для дальнейшего построения ансамбля

```
from __future__ import print_function, division
import caffe
import pandas as pd
import numpy as np
import lmdb

MODEL_PROTO = '/data/deploy.prototxt'
MODEL_WEIGHTS = '/data/_iter_70000.caffemodel'
MEAN = '/data/cifar10zca_mean.binaryproto'

TRAIN_LMDB = '/data/cifar10zca_train_lmdb'
TEST_LMDB = '/data/cifar10zca_test_lmdb'

caffe.set_device(0)
caffe.set_mode_gpu()

# Load CNN
net = caffe.Net(MODEL_PROTO, MODEL_WEIGHTS, caffe.TEST)

# Getting mean
mean_blobproto_new = caffe.proto.caffe_pb2.BlobProto()
f = open(MEAN, 'rb')
mean_blobproto_new.ParseFromString(f.read())
mean_image = caffe.io.blobproto_to_array(mean_blobproto_new)
f.close()

labels = ['airplane', 'automobile', 'bird', 'cat', 'deer',
          'dog', 'frog', 'horse', 'ship','truck']
```

```
29
30 # Preparing database
31 lmdb_cursor = lmdb.open(TRAIN_LMDB).begin().cursor()
32
33 answers = []
34 guesed = 0
35 for i, key, value in enumerate(lmdb_cursor):
36     datum = caffe.proto.caffe_pb2.Datum()
37     datum.ParseFromString(value)
38     label = int(datum.label)
39     image = caffe.io.datum_to_array(datum).astype(np.uint8)
40     image = np.asarray([image]) - mean_image
41     image = image[:, :, 4:36, 4:36]
42
43     forward_pass = net.forward_all(data=image)
44     probabilities = forward_pass['prob'][0]
45     answers.append(np.append(probabilities, label))
46
47     predicted = probabilities.argmax()
48     guesed += int(label == predicted)
49
50 print('Model accuracy: ', guesed / i)
51
52 # Save model answers to csv
53 model = pd.DataFrame(answers, columns=labels + ['ground truth'])
54 model.to_csv('model_train_answers.csv', index=False)
```

Листинг 4 — Построение ансамбля из таблицы ответов нейронных сетей

```
1 import numpy as np
2 import pandas as pd
3 from sklearn.svm import SVC
4 from sklearn.ensemble import ExtraTreesClassifier
5
6 train_data = pd.read_csv('full_train_table.csv').as_matrix()
7 test_data = pd.read_csv('full_test_table.csv').as_matrix()
8
9 # Training set
10 X, y = train_data[:,:-1], train_data[:,-1]
11 # Testing set
12 Xt, yt = test_data[:,:-1], test_data[:,-1]
13
14 # Trying SVM
15 svm = SVC()
16 svm.fit(X,y)
17 svm.score(Xt, yt) * 100
18 >>> 94.170000000000002
```

```
19
20 # Trying Extra Trees
21 etc = ExtraTreesClassifier(n_estimators=1500, max_depth=None,
22                            max_features=4, min_samples_split=1)
23 etc.fit(X,y)
24 etc.score(Xt, yt) * 100
25 >>> 94.210000000000008
26
27 # Write ensemble to filesystem
28 file_path = open('extraTreesEnsemble.pickle', 'wb')
29 pickle.dump(etc, file_path)
30 file_path.close()
```