Оглавление

[Введение 2](#__RefHeading___Toc6844_2443554181)

[1. Набор данных на основе коэффициентов Фурье 4](#__RefHeading___Toc251_201340950)

[2. Искусственные нейронные сети (ИНС) 11](#__RefHeading___Toc257_201340950)

[2.1. Теоретические сведения о ИНС 11](#__RefHeading___Toc259_201340950)

[2.2. Аппаратное и программное обеспечение 17](#__RefHeading___Toc261_201340950)

[2.3. Программная реализация 23](#__RefHeading___Toc1179_1659503810)

[2.4. Модели искусственных нейронных сетей и результаты обучнеия 29](#__RefHeading___Toc1939_3035835693)

[3. Анализ эффективности при использовании коэффициентов Фурье 35](#__RefHeading___Toc265_201340950)

[Список литературы 35](#__RefHeading___Toc625_599451688)

[Приложение 35](#__RefHeading___Toc627_599451688)

[Листинг 1 35](#__RefHeading___Toc629_599451688)

[Листинг 2 36](#__RefHeading___Toc1941_3035835693)

# Введение

На сегодняшний день одним из востребованных, перспективных и интересных направлений является машинное обучение. В частности искусственные нейронные сети (ИНС) которые являются полноценным направлением машинного обучения.

В работе рассматривается что собой представляют коэффициенты Фурье, их вычисление, целесообразность их применения в качестве данных для ИНС, эффективность ИНС при использовании такого набора. Так же в работе затронуты такие проблемы как необходимость наличия оборудования с определенными техническими характеристиками.

Отметим, что даже для входного этапа освоения нейронных сетей необходима сравнительно большая вычислительная мощность. При переходе к реальным задачам требования к вычислительных ресурсам существенно возрастут.

Целью работы является исследование алгоритмов, которые могут уменьшить объем входных данных для нейронной сети без существенной  
потери качества распознавания. Для сокращения входных данных мы переходим от начальной матрицы к некоторым её коэффициентам Фурье.

Важнейшей проблемой является точность распознавания объектов при переходе к коэффициентам Фурье и уменьшении входных данных. Данная проблема является основной в исследовательской работе и занимает большую часть в проведенных изысканий. Поскольку основной идеей и соответственно задачей является попытаться выработать лучшую конфигурацию искусственной нейронной сети с применение коэффициентов Фурье когда при уменьшении размерности входных данных уровень точности не будет падать а в наших мечтах еще и расти. Для достижения поставленной цели были произведены следующие шаги:

1. Получение первоначальных изображений и вычисление коэффициентов Фурье на их основе. Для этого на данном этапе потребовалось изучить некоторые функции из библиотеки numpy. Данные библиотеки пригодятся и в дальнейшей работе.
2. Установка и настройка пакета TensorFlow. Проблематичность этого пункта заключается в том что из-за технических ограничений возможным является использование только версии TensorFlow 1.5. В связи с этим необходимо было установить так же соответствующие версии зависимых пакетов.
3. Написание нескольких различных моделей архитектуры нейронных сетей. Их обучение и сравнение по точности распознавания.
4. Обучение ИНС на минимальном количестве коэффициентов для выбора трех лучших.
5. На выбранных ИНС провести обучение с увеличением количества коэффициентов.
6. Изменение конфигураций моделей архитектуры нейронной сети для увеличения точности распознавания.
7. Анализ полученных данных и формирование выводов о результатах проделанной работы.

Благодаря выполнению всех описанных шагов были получены навыки и знания такие как написание программ для ЭВМ с использование языка программирования питон, знание связанные с машинным обучением и в частности с нейронными сетями, работа и обработка большого количество данных разного типа с использованием языка Pyhton. Однако в первую очередь проделанная работа дает возможность понимания о том как можно сохранить точность распознавания при уменьшении размерности входящих данных.

# Набор данных на основе коэффициентов Фурье

Дискретное преобразование Фурье — это одно из преобразований Фурье, широко применяемых в алгоритмах цифровой обработки сигналов (его модификации применяются в сжатии звука в MP3, сжатии изображений в JPEG и др.), а также в других областях, связанных с анализом частот в дискретном сигнале. Дискретное преобразование - это преобразование, входные и выходные значения которого представляют собой дискретные выборки, что делает его удобным для компьютерных манипуляций.

Прямое ДПФ определяется формулой (1.1)

(1.1)

где N-количество значений сигнала, измеренных за период, а также количество компонент разложения;

при в остальных случаях

Обратное преобразование задается формулой (1.2)

(1.2)

Основная идея преобразования Фурье — это то, что любая картинка может быть представлена в виде суммы синусов и косинусов. Частоты таких слагаемых и будут характеризовать изображение. Для каждой картинки мы говорим, какая из базисных частот в ней преобладает.

Что мы можем сказать о коэффициентах базисных функций? Если у нас большой коэффициент перед базисной функцией с высокой частотой, это означает, что яркость изменяется достаточно часто. На картинке очень много перепадов яркости в маленьких локальных регионах. Если картинка описывается плавными синусоидами, с низкой частотой, то это означает, что на картинке много однородных областей, яркость изменяется плавно, или картинка, например, была «забурена». Таким образом, можно использовать отображение в частотной области для описания изображений.

Для вычисления коэффициентов Фурье в работе используется функция dctn из пакете scipy.fftpack. Которая на вход получает матрицу а возвращает многомерное дискретное косинусное преобразование вдоль осей. В качестве примера возьмем первое изображение из набора Cifar10 представлено на рисунке 1.1 и для простоты записи и компактности изменим размер до 8 на 8 так же сделаем его серым рисунке 1.2 и к полученному изображению уже применим преобразование

Рисунок 1.1 Жаба 28\*28 Рисунок 1.2 Жаба 8\*8 серая

Матрица исходного изображение жаба 8\*8 серая выглядит следующим образом

После применения преобразования Фурье получается матрица (значения округлены до целых).

Теперь применим обратное преобразование что бы убедится что длина векторов и изображение стало изначальным или близкое к изначальному. После применения обратного преобразования получаем матрицу округленную до целых

Для сравнения насколько матрица после обратного преобразования отличается от изначального изображения просто поэлементно вычтем из матрицы начального изображения матрицу после обратного преобразования. В итого получается нулевая матрица.

Для всех тестовых вычислений использовался следующий код

import cv2

import numpy as np

from scipy.fftpack import dctn,idctn

image = cv2.imread('test.png')#загрузка изображения

temp = cv2.cvtColor ( image, cv2.COLOR\_BGR2GRAY )#Преобразование в серый

DCT=dctn(temp,norm='ortho')#Прямое ДПФ

IDCT=idctn(DCT,norm='ortho')#Обратное ДПФ

IDCT=IDCT.astype(int)#округление до целых

t=temp-IDCT#Поэлементное вычитание из матрицы начального изображения матрицы обратного преобразования

#Сохранение результатов

np.savetxt("savepic.txt",temp,fmt='%.d',delimiter='#',newline='\n')

np.savetxt("saveDCT.txt",DCT,fmt='%.d',delimiter='#',newline='\n')

np.savetxt("saveIDCT.txt",IDCT,fmt='%.d',delimiter='#',newline='\n')

np.savetxt("saveP\_I.txt",t,fmt='%.d',delimiter='#',newline='\n')

А для того что бы произвести вычисления и сохранение полученных коэффициентов Фурье набора Cifar10 была реализована программа на python полный код программы листинг 1. Ниже подробно описаны функции используемые в программе.

Функция load загружает исходные данные из файла npz и возвращает два массива CIFAR10['data'] имеет размер 60 000 на 3072, содержит данные о цветовой яркости пикселей картинок из набора cifar10 где одной картинке соответствует один массив размера 3072 где первые 1024 значения отвечают за красный канал вторые 1024 за зеленый а третьи 1024 за синий. В CIFAR10['labels'] хранится информация о том к какому классу относится картинка и он имеет размер 60 000 на 1.

def load():

CIFAR10=np.load('CIFAR10.npz')

return CIFAR10['data'],CIFAR10['labels']

Функция cof принимает массив x возвращает матрицу размером 60 000,3,32,32 которая содержит вычисленные коэффициенты Фурье для картинок из набора Cifar10.

В данной функции создается массив а для хранения вычисленных значений. После в цикле for от 0 до длина массива x -1, т.е мы пробегаем по всем картинкам из набора. Во вложенном цикле for от 0 до 2, поскольку у нас три канала. Вычисляем коэффициенты Фурье по очереди для каждого из трех каналов и записываем их в массив a который после возвращаем.

def cof(x):

a=np.zeros((60000,3,32,32))

x=np.reshape(x,(60000,3,32,32))

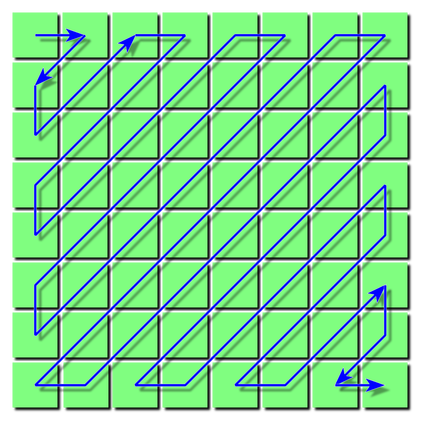
for i in range(len(x)):

for j in range(3):

a[i][j]=dctn(x[i][j])

return a

Функция zig\_zag служит для того что бы взять необходимое количество элементов зиг-загом см рисунок №1.3 поскольку необходимые для работы коэффициент расположены в левом верхнем углу. Функция принимает матрицу из которой необходимо извлечь элементы и число n количество элементов которое нужно взять. Алгоритм прост объявляются два массива b и zigzag который и будет возвращен. Далее в цикле while пока длинна zigzag меньше n. Есть два цикла первый для того что бы взять элементы выше побочной диагонали оригинального массива а второй что бы ниже. В for мы брем срез массива который по сути представляет собой минор из него в список b записываем значения побочной диагонали минора и поверяем если i четно мы разворачиваем b если нет оставляем как есть, в любом случае записываем b в список zigzag. Аналогично устроен и второй цикл for. После список преобразуем к массиву numpy и возвращаем срез из n элементов.

Рисунок №1.3 Обход матрицы зигзагом

def zig\_zag(matrix,n):

b=[]

zigzag=[]

while len(zig\_zag)<n:

for i in range(1,len(matrix)+1):

slice=matrix[:i,:i]

b=[slice[j][i-1-j] for j in range(i)]

if i%2!=0:b.reverse()

zigzag+=b

for i in range(1,len(matrix)):

slice=matrix[i:,i:]

b=[slice[j][len(slice)-1-j] for j in range(len(slice))]

if i%2!=0:b.reverse()

zigzag+=b

np.array(zigzag)

return zigzag[:n]

Функция get\_n\_cof принимает два аргумента, массив а и число n количество элементов которое нужно взять. Возвращает массив в котором содержатся n элементов из трех каналов хранящихся последовательно в одном массиве. get\_n массив который нам будет возвращен. В цикле последовательно пробегаем все строки массива а во вложенном пробегаем все каналы и записываем в get\_n заданное количество элементов взятых зигзагом при помощи функции zig\_zag. В конце возвращаем массив преобразованный к размеру 60 000 на 3\*n, тем самым из трех каналов делаем один поскольку нейронной сети не важно сколько у нас каналов.

def get\_n\_cof(a,n):

get\_n=np.zeros((60000,3,n))

for i in range(len(a)):

if i%100==0:print(i)

for j in range(len(a[i])):

get\_n[i][j]=zig\_zag(a[i][j],n)

return np.reshape(get\_n,(60000,n\*3))

В теле программы мы задаем n количество элементов которое мы хотим взять. Загружаем данные при помощи load в x,y. Вычисляем коэффициенты Фурье используя cof и записываем в А. Задействовав get\_n\_cof берем n элементов и записываем их в Q. После Q сохраняем в npz файл а y в котором содержатся метки в labels.txt.

if \_\_name\_\_ == '\_\_main\_\_':

n=36

x,y=load()

A = cof(x)

Q=get\_n\_cof(A,n)

np.savez('CIFAR10\_cof\_%d.npz'%(n),data=Q)

np.savetxt('labels.txt', y )

После применения данной программы получаем npz файл в котором содержатся некоторое количество коэффициентов Фурье взятое из трех каналов то есть если n=36 то алгоритм возьмёт 36 коэффициентов из красного канала 36 из зеленого и 36 из синего и запишет их в одну строку последовательно получая строку длинной 3\*n. И файл txt с метками который можно создать один раз и использовать в дальнейшем. Поскольку порядок картинок не меняется. Теперь когда имеется необходимые данные для работы можно приступить к написанию моделей ИНС.

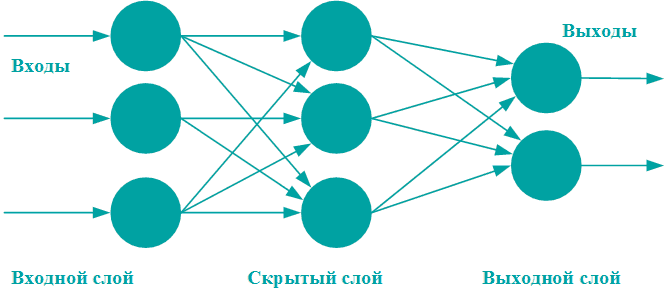
# Искусственные нейронные сети (ИНС)

## Теоретические сведения о ИНС

Перед написанием модели искусственной нейронной сети необходимо разобраться как она в челом устроенна. Принимая в учет то факт что по ИНС написана уже довольно приличное количество книг и различных статей в данной работе сведения будут описаны не детально в кратком виде, который позволит понимать что происходит при этом не усложняя текст.

В начале опишем что такое искусственная нейронная сеть. ИНС это математическая или компьютерная модель восприятия информации мозгом. ИНС имеет несколько структуру представляющую собой связь нескольких ее слоев.

Основные слоя нейронной сеть бывают трех типов входной слой, выходной и скрытые вычислительные. На входы предаются данные по которым требуется получить ответ. Далее они перепродаются в скрытый слой где к ним применяются веса признаков при помощи функции активации. После все передается в выходной слой который формирует выходные данные. Правильность ответа зависит от того насколько хорошо обучена нейронная сеть. Структура ИНС представлена на рисунке 2.1

Рисунок 2.1 Структура искусственной нейронной сети

Прежде чем описывать элементы слоев и модели ИНС опишем что значит о**бучить искусственную нейронную сеть. Под обучением подразумевается что необходимо оптимизировать некоторую целевую функцию так что бы результат работы сети был верен как можно чаще. Говоря об обучении выделяют два основных метода это обучение с учителем и без учителя.**

**Обучение с учителем на вход подается набор тренировочных данных, который обычно называют обучающим. Цель состоит в том что бы уже известные ответы применить но новые данные выраженные виде тестового набора. Необходимо отметить что данные при таком обучении берутся размеченные т. е. в наборе содержаться какие то признаки и к ним есть ответ. Основная суть в данном методе в том что предполагается что данные на которых обучается ИНС похожи на те к которым она будет применяться. В этом методе выделяют две основных задачи классификации и регрессии.**

* **Задача классификации как можно понять из названия тут необходимо определить к какому классу относятся поданные на вход данные.**
* **Задача регрессии состоит в том что требуется предсказать значение некоторой функции, у которая может принимать бесконечно много значений.**

**Обучение без учителя применяется когда у нас есть некоторые данные без разметки и необходимо найти в них какой то смысл. Данный метод применяется для задач кластеризации, снижения размерности, оценки плотности. То есть данный метод применяется в случаях где общей задачей стоит выделение признаков.**

**Стоит так же отметить что выше писанные методы могут «сливаться» и образуют еще различные методы обучения такие как**

* **обучение с частичным привлечением учителя. Когда имеется малое количество размеченных данных и большое не размеченых.**
* **обучение с подкреплением когда за правильное действие алгоритм получает условную «награду»**

**В данной работе была применялось обучение с учителем поскольку данные размечены и перед искусственной нейронной сетью стоит задача классификации изображения. Но распределить по классам необходимо по коэффициентам Фуре.**

**Теперь с пониманием какие есть методы обучения и что вообще значит обучить искусственную** нейронную сеть перейдем к описания слоев.

Как ранее было сказано слоя делятся на три типа входной, выходной и скрытый. Каждый из слоев имеет вход и выход.

* Входной слой его выход равен входу, данный слой не имеет функции активации и ему на вход подаются данные для распознавания. Применяется для передачи данных в ИНС.
* Скрытые слой так же называемый вычислительным. На вход ему передаются данные из предыдущего слоя. К этим данным применяется функция активации и из ее результата и состоит выход.
* Выходной слой является последним слоем искусственной нейронной сети в случае распознавания его задача вернуть нам вектор вероятностей принадлежности к классу. На вход подаются данные из предыдущего слоя к ним применяется некоторая результирующая функция, в данной работе softmax, и на выходе мы получаем вероятности классов.

Как же это работает на вход подаются данные для работы. Входной слой передает их в скрытый где они перемножаются на веса и складываются со смещением а после к этому применяется функция активации и полученные значения передаются дальше. Выходной слой получив информацию из скрытого выдает вероятность принадлежности к классу.

Далее опишем что такое ранее упомянутые веса, смещения и функции активации.

weight(веса) это числовые значения которые определяются в процессе обучения искусственной нейронной сети. Они отвечают за то насколько значим для ИНС параметр.

Bias (смещение) его суть заключается в добавлении константы к входному значению каждого нейрона, что позволяет моделировать некоторую «субъективность» или предубеждение при принятии решения. Bias в нейронных сетях используется для уточнения или изменения выходного значения нейрона.

Функция активации (возбуждения нейрона) определяет выходное значение нейрона в зависимости от результата взвешенной суммы входов и весов. Эти функции обладают следующими свойствами:

1. Ограниченность
2. Стремление к нулю при и к 1 при
3. Везде дифференцируема
4. Производную не сложно вычислить по формуле

В таблице приведены функции возбуждения нейрона используемые в работе.

Таблица 1.Функции активации

|  |  |  |
| --- | --- | --- |
| Название функции | Формула | Производная |
| Логистический сигмоид σ |  |  |
| Гиперболический тангенс tanh |  |  |
| SoftPlus |  |  |
| ReLU |  |  |

В выше приведенной таблице описаны не все функции активации доступные для использования. Конкретно это выбраны поскольку являются основными а те что не указаны по большей части представляют собой их модификации.

Теперь когда описаны основные элементы ИНС перейдем к тому как же все таки обучается она. Как было сказано ранее под методом обучения в искусственных нейронных сетях является уменьшение некоторой заданной функции ошибки. Это означает что мы должны решить задачу оптимизации то есть по заданной функции найти аргументы, в которых эта функция максимизируется или минимизируется. Для этого применяются методы на основе градиентного спуска. В контексте нейронных сетей их так же называют оптимизаторами . Поскольку время на проведение работ ограничено поэтому были использованы только три оптимизатора

* tf.train.GradientDescentOptimizer(Скорость обучения).minimize(Функция ошибки)етодов с накоплением импульса до очевидности проста: «Если мы некоторое время движемся в определённом направлении, то, вероятно, нам следует туда двигаться некоторое время и в будущем»
* tf.train.AdamOptimizer(Скорость обучения).minimize(Функция ошибки). Он сочетает в себе и идею накопления движения и идею более слабого обновления весов для типичных признаков.
* tf.train.AdagradOptimizer( Скорость обучения ).minimize( Функция ошибки ) . Алгоритм адаптивного градиентного спуска он использует разные скорости обучения для каждой итерации. Изменение скорости обучения зависит от разницы параметров во время обучения. Чем больше изменяются параметры, тем менее заметны изменения скорости обучения. Несправедливо иметь одинаковое значение скорости обучения для всех функций. Преимущество использования AdaGrad заключается в том, что он устраняет необходимость вручную изменять скорость обучения. Он более надежен, чем алгоритмы градиентного спуска и их варианты, и достигает сходимости на более высокой скорости. Одним из недостатков AdaGrad является то, что он агрессивно и монотонно снижает скорость обучения. Нередко настает момент, когда скорость обучения становится чрезвычайно низкой. Модель в конечном итоге становится неспособной получить больше знаний, и, следовательно, точность модели ставится под угрозу.

Каждый из них имеет два параметра :

Каждый из них имеет два параметра :

* Скорость обучения показывает, насколько сильно мы сдвигаем параметры в сторону очередного градиента. Скорость обучения — это чрезвычайно важный параметр. Если она будет слишком большой, то алгоритм станет просто прыгать по фактически случайным точкам пространства и никогда не попадет в минимум, потому что все время будет через него перепрыгивать. А если она будет слишком маленькой, то такой проблемы не будет, но, во-первых, обучение станет гораздо медленнее, а во-вторых, алгоритм рискует успокоиться и сойтись в первом же локальном минимуме, который вряд ли окажется самым лучшим.
* Функция ошибки. Это параметр где указывается какую функцию нужно минимизировать либо максимизировать.

Единственное о чем еще не сказано это функция ошибки. Она представляет собой насколько ответ полученный в результате работы нейронной сети верен. Существуют различные функции ошибок MSE «Средняя квадратическая ошибка», MAE «Средняя абсолютная ошибка», RSS «Метод наименьших квадратов» и многие другие. В работе для оценки ответа ИНС используется перекрестная энтропия. Она определяется по формуле

где p — истинное распределение а q- распределение вероятностей.

А для бинарной классификации которую нужно минимизировать на наборе данных выглядит как средняя перекрестная энтропия по всем точкам в данных и определяется формулой

гдеоценка вероятности ответа 1, полученная классификатором.

В результате получается непрерывная функция от от предсказанных классификатором вероятностей, оценивающая насколько качественно он предсказывает метки на данных. может служить функцией ошибки поскольку её можно дифференцировать и оптимизация не вызывает проблем.

## Аппаратное и программное обеспечение

После ознакомления с теоретическими основами можно преступить к практической реализации. Для этой цели используется язык прогромирования Python и его библиотека разработанная специально для создания искусственных нейронных сетей TensorFlow. Помимо этого библиотека numpy.

Python выбран поскольку это язык программирования который широко используется в интернет-приложениях, разработке программного обеспечения, науке о данных и машинном обучении (ML). Разработчики используют Python, потому что он эффективен, прост в изучении и работает на разных платформах. Язык Python имеет следующие преимущества:

* Разработчики могут легко читать и понимать программы на Python, поскольку язык имеет базовый синтаксис, похожий на синтаксис английского.
* Python помогает разработчикам быть более продуктивными, поскольку они могут писать программы на Python, используя меньше строк кода, чем в других языках.
* Python имеет большую стандартную библиотеку, содержащую многократно используемые коды практически для любой задачи. В результате разработчикам не требуется писать код с нуля.
* Разработчики могут легко сочетать Python с другими популярными языками программирования: Java, C и C++.
* Активное сообщество Python состоит из миллионов поддерживающих разработчиков со всего мира. При возникновении проблем сообщество поможет в их решении.
* Кроме того, в Интернете доступно множество полезных ресурсов для изучения Python. Например, вы можете легко найти видеоролики, учебные пособия, документацию и руководства для разработчиков.
* Python можно переносить на различные операционные системы: Windows, macOS, Linux и Unix.

TensorFlow — открытая программная библиотека для машинного обучения, разработанная компанией Google для решения задач построения и тренировки нейронной сети с целью автоматического нахождения и классификации образов, достигая качества человеческого восприятия[5]. Применяется как для исследований, так и для разработки собственных продуктов Google. Основной API для работы с библиотекой реализован для Python, также существуют реализации для R, C#, C++, Haskell, Java, Go, JavaScript и Swift. Некоторые особенности TensorFlow:

* Поддерживаются распределенные вычисления, что облегчает управление данными в больших наборах данных.
* Базовая библиотека подходит для разных методов машинного обучения, а не только для глубокого обучения.
* Эффективное управление сложными математическими структурами, такими как n-мерные массивы.

NumPy (сокращенно от Numerical Python)— библиотека с открытым исходным кодом для языка программирования Python. Возможности:

* поддержка многомерных массивов (включая матрицы);
* поддержка высокоуровневых математических функций, предназначенных для работы с многомерными массивами.

Перед последующим описанием проделанных шагов и этапов работы необходимо проговорить об технических характеристиках вычислительной машины на которой производилось вся проделанная работа. Общих объем оперативной памяти составляет 3.4 гб на систему используется от 700мб до 1 гб. Операционная система Lubuntu 20.04. Процессор имеет следующие характеристики.

Архитектура: x86\_64

CPU op-mode(s): 32-bit, 64-bit

Address sizes: 39 bits physical, 48 bits virtual

Порядок байт: Little Endian

CPU(s): 4

On-line CPU(s) list: 0-3

ID прроизводителя: GenuineIntel

Имя модели: Intel(R) Pentium(R) CPU N4200 @ 1.10GHz

Семейство ЦПУ: 6

Модель: 92

Потоков на ядро: 1

Ядер на сокет: 4

Сокетов: 1

Степпинг: 9

CPU max MHz: 2500,0000

CPU min MHz: 800,0000

BogoMIPS: 2188.80

Флаги: fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36 clflush dts acpi mmx fxsr sse sse2 ss ht tm pbe syscall nx pdpe1gb rdtscp lm constant\_tsc art arch\_perfmon pebs bts rep\_good nopl xtopology tsc\_reliable nonstop\_tsc cpuid aperfmperf tsc\_known\_freq pni pclmulqdq dtes64 monitor ds\_cpl est tm2 ssse3 sdbg cx16 xtpr pdcm sse4\_1 sse4\_2 x2apic movbe popcnt tsc\_deadline\_timer aes xsave rdrand lahf\_lm 3dnowprefetch cpuid\_fault cat\_l2 ibrs ibpb stibp fsgsbase tsc\_adjust smep erms mpx rdt\_a rdseed smap clflushopt intel\_pt sha\_ni xsaveopt xsavec xgetbv1 xsaves dtherm ida arat pln pts md\_clear arch\_capabilities.

Стоит так же отметить что все программы запускались через терминал а код писался в приложении FeatherPad (легковесный текстовый редактор со вкладками).

Теперь перейдем к программному обеспечению. Для того что бы применить данную библиотеку сперва ее требуется установить. Как и у всех библиотек Python для этого существуют два метода при помощи команды pip install tensorflow или собрать из исходного кода который доступен но оффициальном сайте а так же на GitHub.

В данной работе использовался метод через pip. Но при кажущейся простоте тут есть некоторые количество подводных камней.

Первая через pip устанавливается самая последняя версия библиотеки. Тут и возникает проблема что процессор не поддерживает необходимые инструкции которые требуются процессору.

Вторая подобрать и установить версию совместимую с процессором. Если с подбором не возникло ни каких сложностей и ответ в интернете нашелся относительно быстро то с установкой возникли проблемы. Для установки Tensorflow версии совместимой с ЭВМ требуется питон ниже 3.7 в системе установлена python3.10.12. Для решения этой проблемы использовалась утилита pyenv 2.3.31. pyenv позволяет вам легко переключаться между несколькими версиями Python. Это просто, ненавязчиво и следует традиции UNIX - универсальных инструментов, которые хорошо делают что-то одно.

Чтобы установить данный инструмент требуется ввести

apt-get install -y make build-essential libssl-dev zlib1g-dev \

libbz2-dev libreadline-dev libsqlite3-dev wget curl llvm libncurses5-dev \

libncursesw5-dev xz-utils tk-dev libffi-dev liblzma-dev python-openssl тем самым установить требуемые пакеты. После установки требуется не большая настройка для окружения в файл ~/.bashrc необходимо добавить следующие строки и для применения изменения использовать команду source ~/.profile.

export PYENV\_ROOT="${HOME}/.pyenv"

if [ -d "${PYENV\_ROOT}" ]; then

export PATH="${PYENV\_ROOT}/bin:${PATH}"

eval "$(pyenv init -)"

eval "$(pyenv virtualenv-init -)"

fi

Теперь инструмент готов к работе. Устанавливаем в python 3.6.15 командой pyenv install 3.6 теперь он доступен для использования но не включен по умолчанию, скачевается и устанавливается последняя версия из релиза но при необходимости можно указать и другую. Создаем папку где будет располагаться VENV открываем терминал и переходим в расположении созданной папки и прописываем pyenv local 3.6.15 данная команда устанавливает что в этой директории используется питон версии 3.6.15.

Теперь при наличии соответсвующей версии python можно установить tensorflow. Сначала создадим VENV (virtual environment) для того что бы проект имел свои собственные зависимости, независимо от других проектов. Это позволило избежать ситуации когда зависимый пакет установлен в системе более новый а не обходим старый. И заключительный этап обновить pip а после установить tensorflow 1.5 командой pip install tensorflow==1.5 вместе с ним установятся и зависимости. Так же после установки на будущее был создай файл requirements.txt, который содержит все пакеты в venv их список приведен ниже :

1. absl-py==1.4.0
2. bleach==1.5.0
3. dataclasses==0.8
4. html5lib==0.9999999
5. importlib-metadata==4.8.3
6. Markdown==3.3.7
7. numpy==1.19.5
8. protobuf==3.19.6
9. six==1.16.0
10. tensorflow==1.5.0
11. tensorflow-tensorboard==1.5.1
12. typing\_extensions==4.1.1
13. Werkzeug==2.0.3
14. zipp==3.6.0

## Программная реализация

В данной главе описывается общий программный код который больше представляет собой шаблон изменяемый под конкретную модель. Полный листинг программы смотри листинг 2

Функция load служит для загрузки данных она возвращает два массива numpy. CIFAR10['data'] содержит набор коэффициентов Фурье размеров 60 000 на 3\*N, N число взятых коэффициентов из каждого канала RGB. В конкретном случае 36 следовательно размер массива равен 60 000 на 128. CIFAR\_LABELS имеет размер 60 000 на 1 и содержит ответы к набору. Т.е. к какому классу относится изображение.

def load():

CIFAR10=np.load('CIFAR10\_cof\_36\_DCT.npz')

CIFAR\_LABELS=np.loadtxt("labels.txt")

return CIFAR10['data'],CIFAR\_LABELS

Функция nextBatch применяется для взятия следующего минибатча из обучающих данных. Она принимает два параметра размер выборки которую мы хотим взять и текущую итерацию цикла. i0 индекс начала минибатча i1 конец минибатча. Тут мы проверяем если конец выходит за пределы размерности наших тренировочных данных мы устанавливаем начало батча в начало тренировочных данных а конец равный размеру батча. И функция нам возвращает два среза массивов первый с коэффициентами для тренировки а второй с ответами.

def nextBatch(bSize, i):

i0 = (i\*bSize)%my

i1 = i0+bSize

if( i1>my ):

i0=0

i1=bSize

return X\_train[i0:i1,:], y10[i0:i1,:]

Функция fullyconnected\_layer() необходима для объявления полносвязного слоя. Имеет три параметра предыдущий тензор, input\_size размер входных данных, т. е. выходные значения предыдущего слоя, out\_size размер выхода этого слоя. Возвращает значения после применения функции активации. В коде четыре return служат только для субъективного удобства что бы оставить один с необходимой функцией возбуждения остальные закоментировать. В шаблоне и имеются еще две подобных функции fullyconnected\_layer1(), fullyconnected\_layer2() устроены аналогично описанной и такое разделение только для собственного удобства. Так же стоит заметить что в для хорошего обучения искусственной нейронной сети необходимо хорошо подобрать начальную инициализацию весов. В работе это происходит при инициализации через функцию truncated\_normal в ней главным образом для работы важен параметр stddev который отвечает за стандартное отклонение нормального распределения перед усечением. И как показала практика возьмешь слишком большой обучение может застрять где то в начале и точность не превысит 15%. С слишком маленьким значением ситуация аналогичная. Главная проблема в том что нет ни каких рекомендация по этому и приходится подбирать значение просто подбором.

def fullyconnected\_layer(tensor, input\_size, out\_size):

W = tf.Variable(tf.truncated\_normal([input\_size, out\_size], stddev=0.001302313))

b = tf.Variable(tf.truncated\_normal([out\_size], stddev=0.001302313))

return tf.nn.relu(tf.matmul(tensor, W) + b)

#return tf.nn.sigmoid(tf.matmul(tensor, W) + b)

#return tf.nn.softplus(tf.matmul(tensor, W) + b)

#return tf.nn.tanh(tf.matmul(tensor, W) + b)

train\_size задает размер тренировочного набора а конкретно во всех случаях составляет 85%. В X,Y загружаются полные наборы в X набор коэффициентов в Y ответы к ним каждый содержит 60 000 строк. X\_train тренировочные данные получаются срезом из X от 0 до train\_size. X\_test тестовые данные получаются срезом из X от train\_size до конца. Y\_train ответы к тренировочным данным получаются срезом из Y от 0 до train\_size. Y\_test ответы к тестовым данным получаются срезом из Y от train\_size до конца.y10 представляет собой преобразованные ответы для тренировки из числа в вектор . К примеру если ответ был 6 то получиться вектор длинны 10 равный [0,0,0,0,0,1,0,0,0,0]. y10t аналогично y10 но для тестового.

train\_size = int(0.85 \* 60000)

X,Y=load()

X\_train=X[:train\_size]

X\_test=X[train\_size:]

Y\_train=Y[:train\_size]

Y\_test=Y[train\_size:]

Y\_train=Y\_train.astype(int)

Y\_test=Y\_test.astype(int)

y10=np.eye(10)[Y\_train]

y10t=np.eye(10)[Y\_test]

#------------------------------------------------------------------------------------------------------------------

my = Y\_train.shape[0]# Определение размера данных для обучения

Nin = X\_train.shape[1]# Определение количества входных параметров

Nout = 10 # Количество выходов

learning\_rate=0.00002533# Скорость обучения

Batchsize=613# Размер минибатча

learning\_time=100000 # Время обучения

#------------------------------------------------------------------------------------------------------------------

цикл for служит что бы проверить как влияет увеличение количества нейронов на точность распознавания.

for w in range(3):

N0 =int((Nin\*(w+2))/2)#количество нейронов в первом слое

N1 = int(N0/1.5)#количество нейронов во втором слое

N2=int(N1/2)#количество нейронов в третьем слое

print(Nin,N0,N1,N2)

'''задается имя файла в который будет сохранен результат обучения '''

file='Result/RRDCT\_N0\_%d\_N1\_%d\_N2\_%d.txt'%(N0,N1,N2)

'''задается имя файла в котором сохраняется точность во время обучения'''

file1='Result/validation\_RRDCT\_N0\_%d\_N1\_%d\_N2\_%d.txt'%(N0,N1,N2)

#------------------------------------------------------------------------------------------------------------------

'''Описание модели'''

x = tf.placeholder(tf.float32, [None, Nin])#Входной слой

L1=fullyconnected\_layer(x,Nin,N0)#первый скрытый слой

L2=fullyconnected\_layer1(L1,N0,N1)#Второй скрытый слой

L3=fullyconnected\_layer2(L2,N1,N2)#Третий скрытый слой

#------------------------------------------------------------------------------------------------------------------

#Слой дропаута

keep\_probability=tf.placeholder(tf.float32)

ld=tf.nn.dropout(l2,keep\_probability)

#------------------------------------------------------------------------------------------------------------------

Выходной совтмакс слой

W\_out = tf.Variable(tf.zeros([N2, Nout]))

b\_out = tf.Variable(tf.zeros([Nout]))

y=tf.nn.softmax(tf.matmul(L3,W\_out)+b\_out)

#------------------------------------------------------------------------------------------------------------------

y\_ = tf.placeholder(tf.float32, [None, Nout])

correct\_prediction = tf.equal(tf.argmax(y, 1), tf.argmax(y\_, 1))#Определение верности предсказания.

accuracy = tf.reduce\_mean(tf.cast(correct\_prediction, tf.float32))#Определение точности предсказания

#------------------------------------------------------------------------------------------------------------------

'''Описание функции ошибки'''

logit=tf.matmul(L3,W\_out)+b\_out

cross\_entropy= tf.reduce\_mean(tf.nn.softmax\_cross\_entropy\_with\_logits(logits=logit,labels= y\_))

#------------------------------------------------------------------------------------------------------------------

'''Выбор оптимизатора при обучении не нужные закомментировать необходимый оставить'''

#train\_step = tf.train.AdagradOptimizer(learning\_rate).minimize(cross\_entropy)

train\_step = tf.train.AdamOptimizer(learning\_rate).minimize(cross\_entropy)

#train\_step = tf.train.GradientDescentOptimizer(learning\_rate).minimize(cross\_entropy)

optimizer='Adam'

#------------------------------------------------------------------------------------------------------------------

Цикл For для того что бы каждая конфигурации сделала n количество прогонов

for q in range(3):

start=time.time()#время начала прохода, для определения времени работы в минутах

#------------------------------------------------------------------------------------------------------------------

''' Сессия обучения заданной модели искусственной нейронной сети'''

with tf.Session() as sess:

sess.run(tf.global\_variables\_initializer())

print("Start!-----------------")

for i in range(learning\_time):

if (i+1)%5000==0:

'''Вывод эпохи и текущей точности на валидационом множестве'''

print('ep %5d: %6d: %7.5f'%((i+1)/100, i, sess.run(accuracy, feed\_dict={x: X\_train, y\_: y10})))

if (i+1)%10000==0:

f = open(file1,'a')

f.write(str(i)+"\t"+str(sess.run(accuracy, feed\_dict={x: X\_train, y\_: y10}))+"\n") #Запись текущей точности в файл

f.close()

batch\_xs, batch\_ys = nextBatch(Batchsize, i)

sess.run(train\_step, feed\_dict={x: batch\_xs,y\_:batch\_ys})

er=sess.run(accuracy, feed\_dict={x: X\_test, y\_: y10t} )# конечная точность после обучения

f = open(file,'a')

'''Запись в файл номер прохода, конечную точность, размер выборки, время обучения, скорость обучения, оптимизатор.'''

f.write(f"%d \t error = %f \t batch size = %d \t learning time = %d \t learning rate = %f \t optimaizer = %s \t "%(q,er,Batchsize,learning\_time,learning\_rate,optimizer))

f.close()

print("Точность: %s",er)

end=time.time()# Конец одного прохода

f = open(file,'a')

f.write('time = '+str((end-start)/60)+'\n') # Запись времени одного прохода

f.close()

print('-----End----')

Данный шаблон позволяет легко менять параметры модели что упрощает работу. Теперь перейдем к описания моделей ИНС которые были реализованы при помощи шаболна.

## Модели искусственных нейронных сетей и результаты обучнеия

В работе реализованы и обучены несколько моделей ИНС одни с 2 скрытыми слоями а другие с 3. В начале работы для оптимизации реализованы были двухслойные искусственные нейронные сети с различными функциями активации и тремя оптимизаторами. Ниже в таблице № 1.1 приведены конфигурации для слоя 1(L1) и слоя 2(L2) с четырьмя функциями возбуждения нейрона.

Таблица 1.1 Конфигурации функций активации для двух слоев.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| L1/L2 | Relu | Sigmoid | Softplus | Tahn |
| Relu | Relu,Relu | Relu,Sigmoid | Relu,Softplus | Relu,Tahn |
| Sigmoid | Sigmoid,Relu | Sigmoid,Sigmoid | Sigmoid,Softplus | Sigmoid,Tahn |

Продолжение таблицы 1.1

| L1/L2 | Relu | Sigmoid | Softplus | Tahn |
| --- | --- | --- | --- | --- |
| Softplus | Softplus,Relu | Softplus,Sigmoid | Softplus,Softplus | Softplus,Tahn |
| Tahn | Tahn,Relu | Tahn,Sigmoid | Tahn,Softplus | Tahn,Tahn |

Данные конфигурации были обучены на минимальном, по субъективному мнению, количестве коэффициент Фурье а именно на 36 взятых из матрицы зигзагом из каждого канала RGB. После выбрать конфигурации которые показали лучший результат в точности распознавания. Следующим этапом проверить как измениться точность при большем количестве данных. Результаты обучения при использовании оптимизатора Adagrad, временем обучения 60 000, скоростью обучения 0,00013 и при различном количестве нейронов в слоях указаны в таблицах 1.2, 1.3, 1.4.

Таблица 1.2 Результаты при 108 в L1 и 54 в L2

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| L1/L2 | Relu | Sigmoid | Softplus | Tahn |
| Relu | 51,1 | 20,88 | 51,57 | 32,60 |
| Sigmoid | 17,85 | 9,90 | 9,90 | 17,85 |
| Softplus | 51,53 | 22,10 | 51,63 | 31,73 |
| Tahn | 18,97 | 9,90 | 17,70 | 18,07 |

Таблица 1.3 Результаты при 162 в L1 и 81 в L2

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| L1/L2 | Relu | Sigmoid | Softplus | Tahn |
| Relu | 53,27 | 25,90 | 51,00 | 43,23 |
| Sigmoid | 18,45 | 9,90 | 9,90 | 18,03 |
| Softplus | 52,97 | 25,30 | 53,10 | 43,50 |
| Tahn | 20,30 | 9,90 | 17,90 | 18,37 |

Таблица 1.4 Результаты при 216 в L1 и 108 в L2

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| L1/L2 | Relu | Sigmoid | Softplus | Tahn |
| Relu | 54,03 | 34,57 | 52 | 48,10 |

Продолжение таблицы 1.4

| L1/L2 | Relu | Sigmoid | Softplus | Tahn |
| --- | --- | --- | --- | --- |
| Softplus | 53,90 | 34,73 | 54,07 | 47,50 |
| Tahn | 20,70 | 9,90 | 18,80 | 20,63 |

Из таблиц наглядно видно что наилучшие результаты в распознавании

дали следующие конфигурации

1. Relu, Relu. 54% при 216 нейронах в первом слое и 108 во втором
2. Rellu, Softolus 52% при 216 нейронах в первом слое и 108 во втором
3. Relu tahn 48,1% при 216 нейронах в первом слое и 108 во вотором
4. softplus relu53,9% при 216 нейронах в первом слое и 108 во втором
5. softplus softplus 54,07% при 216 нейронах в первом слое и 108 во втором

Поскольку конфигурации Relu, Softplus и Softplus, Relu по сути зеркальные и  Relu, Softplus показала к тому же меньший результат поэтому для дальнейшей работы будет применяется Softplus, Relu.

Теперь будем обучать на большем количестве коэффициентов искусственные нейронные сети со следующими конфигурациями функций возбуждения:

1. Relu, Relu.
2. Relu tahn
3. softplus relu
4. softplus softplus

Но сначала для большей объективности обучим эти конфигурации на все тех же 36 коэффициентах на но дрргих оптимизаторах Adam и GradientDescent, при все той же скорости обучения 0,00013. Результаты обучения для Adam приведены в таблице 1.5 а результаты для GradientDescent в таблице 1.6

Таблца 1.5 Результаты при оптимизаторе Adam.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Количество Нейронов | Relu,Relu | Softplus,Relu | Softplus,Softplus | Relu,Tahn |
| 108,54 | 51,77 | 51,27 | 51,17 |  |
| 162,81 | 49,90 | 48,76 | 48,63 |  |
| 216,108 | 47,97 | 47,07 | 47,10 |  |

Таблица 1.6 Результаты при оптимизаторе GradientDescent.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Количество Нейронов | Relu,Relu | Softplus,Relu | Softplus,Softplus | Relu,Tahn |
| 108,54 | 51,13 | 50,63 | 51,43 |  |
| 162,81 | 50,23 | 49,47 | 50,43 |  |
| 216,108 | 48,23 | 47,77 | 48,65 |  |

Для простоты сравнения выпишем так же результаты при оптимизаторе Adagrad в таблицу 1.7

Таблица 1.7 Результаты при оптимизаторе Adagrad

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Количество Нейронов | Relu,Relu | Softplus,Relu | Softplus,Softplus | Relu,Tahn |
| 108,54 | 50,63 | 51,57 | 51,63 | 32,60 |
| 162,81 | 53,27 | 51,00 | 53,10 | 43,23 |
| 216,108 | 54,03 | 52,00 | 54,07 | 48,10 |

Далее обучение производилось на 128 коэффициентах Фурье взятых из каждого канала RGB. Но теперь кроме оптимизатора Adagrad использовалась Adam, GradientDescent. При скорости обучения 0,00013 и размером минибатча 313, однако в процессе выяснилось что для каждого оптимизатора требуется подбирать свою длительность обучения поскольку при фиксированной в 60 000 происходило переобучение. Из-за этого результаты обучения получились довольно не однородные и по этому составить таблицу не представляется возможным и данные будут подаваться в виде списка. Где первым будет указываться оптимизатор, вторым длительность обучения, третьим точность распознавания в процентах, так же в некоторых случаях размер минибатча который не равен стандартному и скорость обучения если изменялась.

Результаты для конфигурации Relu,Relu следующие для ИНС с 384 нейронами в первом слое и 192 во втором.

1. Adagrad , 40 000, 53,52%
2. Adam. 30 000, 44,60%, при минибатче 613
3. Adam, 30 000, 50,60% при минибатче 213 и скорость обучения менялась от 0,0001 до0,000013 при новом обчучении
4. Gradient, 50 000, 31.9%
5. Gradient, 40 000 , 48,08%

Результаты для конфигурации Relu,Relu следующие для ИНС с 576 нейронами в первом слое и 288 во втором.

1. Gradient, от 30 000 до 40 000 0,09 переобучение
2. Adam, 20 000, 50,1%
3. Adagrad, 50 000, 52,9%

Результаты для конфигурации Relu,Relu следующие для ИНС с 768 нейронами в первом слое и 384 во втором.

1. Gradient. 30 000m 9% переобучение
2. Gradient, 20 000, 50,5%
3. Adagrad, 50 000, 53,8%
4. Adam, 50 000, 49,8%

Результаты для конфигурации Softplus,Relu следующие для ИНС с 384 нейронами в первом слое и 192 во втором.

1. Adagrad , 40 000, 52,80%
2. Adam. 30 000, 48,36%,
3. Gradient, 60 000, 48.15%
4. Gradient, 40 000 , 47,90%

Результаты для конфигурации Softplus,Relu следующие для ИНС с 576 нейронами в первом слое и 288 во втором.

1. Gradient, от 30 000 до 40 000 47,7%
2. Adam, 25 000, 50,06%
3. Adagrad, 45 000, 53,9%

Результаты для конфигурации Softplus,Relu следующие для ИНС с 768 нейронами в первом слое и 384 во втором.

1. Adam, 40 000, 52,86%
2. Adagrad, 40 000, 50,09%

Результаты для конфигурации Softplus, Softplus следующие для ИНС с 384 нейронами в первом слое и 192 во втором.

1. Adagrad , 40 000, 52,80%
2. Adam. 30 000, 48,36%,
3. Gradient,40 000, 47,84%

Результаты для конфигурации Softplus, Softplus следующие для ИНС с 576 нейронами в первом слое и 288 во втором.

1. Gradient, от 30 000, 47,2%
2. Gradient, от 40 000, 37,4%
3. Adam, 50 000, 43,36%
4. Adagrad, 35 000, 52,6%

Результаты для конфигурации Softplus, Softplus следующие для ИНС с 768 нейронами в первом слое и 384 во втором.

1. Adam, 40 000, 50,03%
2. Adagrad, 30 000, 53,04%
3. Gradient, от 20 000, 49,20%
4. Gradient, от 40 000, 9% переобучение

Результаты для конфигурации Relu, Tahn следующие для ИНС с 384 нейронами в первом слое и 192 во втором.

1. Adagrad , 50 000, 47,1%
2. Adam. 60 000, 50,08%,
3. Adam. 50 000, 50,20%,
4. Gradient,40 000, 52,1%
5. Gradient,60 000, 51,72%

Результаты для конфигурации Relu, Tahn следующие для ИНС с 576 нейронами в первом слое и 288 во втором.

1. Gradient, от 60 000, 52,56%
2. Adam, 50 000, 50,01%
3. Adagrad, 45 000, 49,10%

Результаты для конфигурации Relu, Tahn следующие для ИНС с 768 нейронами в первом слое и 384 во втором.

1. Gradient, от 60 000, 53,50%
2. Adam, 60 000, 49,8%
3. Adam, 40 000, 50,23%
4. Adagrad, 40 000, 49,90%

ЕСТЬ ЕЩЕ ДЛЯ 1024ИЗ КАЖЖДОГО МОЖЕТ ЕЩЕ ЧТО ТО ВЫСЧИТАЮ И ВПИШУ НО СЕЙЧАС ВСЕ СПАТЬ

# Анализ эффективности при использовании коэффициентов Фурье

# Список литературы

# Приложение

# Листинг 1

import numpy as np

from scipy.fftpack import dctn

def load():

CIFAR10=np.load('CIFAR10.npz')

return CIFAR10['data'],CIFAR10['labels']

#--------------------------------------------------------------------------------------------------

def cof(x):

a=np.zeros((60000,3,32,32))

x=np.reshape(x,(60000,3,32,32))

for i in range(len(x)):

for j in range(3):

a[i][j]=dctn(x[i][j])

return a

#--------------------------------------------------------------------------------------------------

def zig\_zag(matrix,n):

b=[]

zigzag=[]

while len(zig\_zag)<n:

for i in range(1,len(matrix)+1):

slice=matrix[:i,:i]

b=[slice[j][i-1-j] for j in range(i)]

if i%2!=0:b.reverse()

zigzag+=b

for i in range(1,len(matrix)):

slice=matrix[i:,i:]

b=[slice[j][len(slice)-1-j] for j in range(len(slice))]

if i%2!=0:b.reverse()

zigzag+=b

np.array(zigzag)

return zigzag[:n]

#--------------------------------------------------------------------------------------------------

def get\_n\_cof(a,n):

get\_n=np.zeros((60000,3,n))

for i in range(len(a)):

if i%100==0:print(i)

for j in range(len(a[i])):

get\_n[i][j]=zig\_zag(a[i][j],n)

return np.reshape(get\_n,(60000,n\*3))

#--------------------------------------------------------------------------------------------------

if \_\_name\_\_ == '\_\_main\_\_':

n=36

x,y=load()

A = cof(x)

Q=get\_n\_cof(A,n)

np.savez('CIFAR10\_cof\_%d\_shift.npz'%(n),data=Q)

np.savetxt('labels.txt', y )

# Листинг 2

import numpy as np

import tensorflow as tf

import time

def load():

CIFAR10=np.load('CIFAR10\_cof\_36\_DCT.npz')

CIFAR\_LABELS=np.loadtxt("labels.txt")

return CIFAR10['data'],CIFAR\_LABELS

#------------------------------------------------------------------------------------------------------------------

def nextBatch(bSize, i):

i0 = (i\*bSize)%my

i1 = i0+bSize

if( i1>my ):

i0=0

i1=bSize

return X\_train[i0:i1,:], y10[i0:i1,:]

#------------------------------------------------------------------------------------------------------------------

def fullyconnected\_layer(tensor, input\_size, out\_size):

W = tf.Variable(tf.truncated\_normal([input\_size, out\_size], stddev=0.001302313))

b = tf.Variable(tf.truncated\_normal([out\_size], stddev=0.001302313))

return tf.nn.relu(tf.matmul(tensor, W) + b)

#return tf.nn.sigmoid(tf.matmul(tensor, W) + b)

#return tf.nn.softplus(tf.matmul(tensor, W) + b)

#return tf.nn.tanh(tf.matmul(tensor, W) + b)

#------------------------------------------------------------------------------------------------------------------

def fullyconnected\_layer1(tensor, input\_size, out\_size):

W = tf.Variable(tf.truncated\_normal([input\_size, out\_size], stddev=0.001302313))

b = tf.Variable(tf.truncated\_normal([out\_size], stddev=0.003102313))

return tf.nn.relu(tf.matmul(tensor, W) + b)

#return tf.nn.sigmoid(tf.matmul(tensor, W) + b)

#return tf.nn.softplus(tf.matmul(tensor, W) + b)

#return tf.nn.tanh(tf.matmul(tensor, W) + b)

#------------------------------------------------------------------------------------------------------------------

def fullyconnected\_layer2(tensor, input\_size, out\_size):

W = tf.Variable(tf.truncated\_normal([input\_size, out\_size], stddev=0.001302313))

b = tf.Variable(tf.truncated\_normal([out\_size], stddev=0.003102313))

return tf.nn.relu(tf.matmul(tensor, W) + b)

#return tf.nn.sigmoid(tf.matmul(tensor, W) + b)

#return tf.nn.softplus(tf.matmul(tensor, W) + b)

#return tf.nn.tanh(tf.matmul(tensor, W) + b)

#------------------------------------------------------------------------------------------------------------------

train\_size = int(0.85 \* 60000)

X,Y=load()

X\_train=X[:train\_size]

X\_test=X[train\_size:]

Y\_train=Y[:train\_size]

Y\_test=Y[train\_size:]

Y\_train=Y\_train.astype(int)

Y\_test=Y\_test.astype(int)

y10=np.eye(10)[Y\_train]

y10t=np.eye(10)[Y\_test]

#------------------------------------------------------------------------------------------------------------------

my = Y\_train.shape[0]

Nin = X\_train.shape[1]

Nout = 10

learning\_rate=0.00002533

Batchsize=613

learning\_time=100000

#------------------------------------------------------------------------------------------------------------------

for w in range(3):

N0 =int((Nin\*(w+2))/2)

N1 = int(N0/1.5)

N2=int(N1/2)

print(Nin,N0,N1,N2)

file='Result/11RRDCT\_N0\_%d\_N1\_%d\_N2\_%d.txt'%(N0,N1,N2)

file1='Result/111validation\_RRDCT\_N0\_%d\_N1\_%d\_N2\_%d.txt'%(N0,N1,N2)

#------------------------------------------------------------------------------------------------------------------

'''Описание модели'''

x = tf.placeholder(tf.float32, [None, Nin])

L1=fullyconnected\_layer(x,Nin,N0)

L2=fullyconnected\_layer1(L1,N0,N1)

L3=fullyconnected\_layer2(L2,N1,N2)

W\_out = tf.Variable(tf.zeros([N2, Nout]))

b\_out = tf.Variable(tf.zeros([Nout]))

y=tf.nn.softmax(tf.matmul(L3,W\_out)+b\_out)

y\_ = tf.placeholder(tf.float32, [None, Nout])

correct\_prediction = tf.equal(tf.argmax(y, 1), tf.argmax(y\_, 1))

accuracy = tf.reduce\_mean(tf.cast(correct\_prediction, tf.float32))

logit=tf.matmul(L3,W\_out)+b\_out

cross\_entropy = tf.reduce\_mean(tf.nn.softmax\_cross\_entropy\_with\_logits(logits=logit,labels=y\_))

#train\_step = tf.train.AdagradOptimizer(learning\_rate).minimize(cross\_entropy)

train\_step = tf.train.AdamOptimizer(learning\_rate).minimize(cross\_entropy)

#train\_step = tf.train.GradientDescentOptimizer(learning\_rate).minimize(cross\_entropy)

optimizer='Adam'

#------------------------------------------------------------------------------------------------------------------

for q in range(3):

start=time.time()

with tf.Session() as sess:

sess.run(tf.global\_variables\_initializer())

print("Start!-----------------")

for i in range(learning\_time):

if (i+1)%5000==0:

print('ep %5d: %6d: %7.5f'%((i+1)/100, i, sess.run(accuracy, feed\_dict={x: X\_train, y\_: y10})))

if (i+1)%10000==0:

f = open(file1,'a')

f.write(str(i)+"\t"+str(sess.run(accuracy, feed\_dict={x: X\_train, y\_: y10}))+"\n")

f.close()

batch\_xs, batch\_ys = nextBatch(Batchsize, i)

sess.run(train\_step, feed\_dict={x: batch\_xs,y\_:batch\_ys})

er=sess.run(accuracy, feed\_dict={x: X\_test, y\_: y10t} )

f = open(file,'a')

f.write(f"%d \t error = %f \t batch size = %d \t learning time = %d \t learning rate = %f \t optimaizer = %s \t "%(q,er,Batchsize,learning\_time,learning\_rate,optimizer))

f.close()

print("Точность: %s",er)

end=time.time()

f = open(file,'a')

f.write('time = '+str((end-start)/60)+'\n')

f.close()

print('-----End----')