МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ**

**«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ «МИСиС»**

Институт ИТКН

Кафедра инженерной кибернетики

Направление подготовки: «01.03.04 Прикладная математика»

Квалификация: бакалавр

Группа: БПМ-17-2

**ОТЧЕТ**

**ПО КУРСОВОЙ РАБОТЕ**

**«ИСКУССТВЕННЫЕ НЕЙРОННЫЕ СЕТИ»**

на тему: «Сверточные сети. Сеть Inception»

**Студент** \_\_\_\_\_\_\_\_\_\_/ Долгов А.А.

**Руководитель** \_\_\_\_\_\_\_\_\_\_/ старший преподаватель, Кондыбаева А.Б.

**Оценка: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Дата защиты: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Москва 2020**

**Содержание**

1. Введение 3

2. Теоретическая часть 5

2.1. Нейронные сети 5

2.2. Сверточные нейронные сети 6

2.3. Модель Inception 7

3. Реализация модели 12

3.1. Используемые программные средства 12

3.2. Разработанные классы 12

3.3. Тестирование модели 14

4. Заключение 18

5. Список использованных источников 19

Приложение. 20

1. **Введение**

Глубокие нейронные сети в настоящее время становятся одним из самых популярных методов машинного обучения. Они показывают лучшие результаты по сравнению с альтернативными методами в таких областях, как распознавание речи, обработка естественного языка, компьютерное зрение, медицинская информатика и др. Одна из причин успешного применения глубоких нейронных сетей заключается в том, что сеть автоматически выделяет из данных важные признаки, необходимые для решения задачи. В альтернативных алгоритмах машинного обучения признаки должны выделяться людьми, существует специализированное направление исследований — инженерия признаков (feature engineering). Однако при обработке больших объемов данных нейронная сеть справляется с выделением признаков гораздо лучше, чем человек.

Модель искусственных нейронных сетей была предложена в 1943 году, а сам термин глубокое обучение (deep learning) стал широко использоваться только начиная с 2006 года. До этого применялись термины загрузка глубоких сетей (loading deep networks) и обучение глубокой памяти (learning deep memories). Рост популярности глубоких нейронных сетей, происходящий в последние несколько лет, можно объяснить тремя факторами. Во-первых, произошло существенное увеличение производительности компьютеров, в том числе ускорителей вычислений GPU (Graphics Processing Unit), что позволило обучать глубокие нейронные сети сети значительно быстрее и с более высокой точностью. Ранее имеющихся вычислительных мощностей не хватало для обучения сколько-нибудь сложной сети, пригодной для решения практических задач. Во-вторых, был накоплен большой объем данных, который необходим для обучения глубоких нейронных сетей. В-третьих, разработаны методы обучения нейронных сетей, позволяющие быстро и качественно обучать сети, состоящие из ста и более слоев, что раньше было невозможно из-за проблемы исчезающего градиента и переобучения. Сочетание трех факторов привело к существенному прогрессу в обучении глубоких нейронных сетей и их практическом использовании, что позволило глубоким нейронным сетям занять лидирующую позицию среди методов машинного обучения.

Целью данной курсовой работы является изучение работы сверточной сети модели Inception, а также разработка алгоритма классификации собак и кошек.

**2. Теоретическая часть**

**2.1. Нейронные сети**

Нейронная сеть – это сеть или схема нейронов, или в современном смысле искусственная нейронная сеть, состоящая из искусственных нейронов или узлов. Таким образом, нейронная сеть является либо биологической нейронной сетью, состоящей из реальных биологических нейронов, либо искусственной нейронной сетью, для решения задач искусственного интеллекта (ИИ). Нейронные сети представляют собой набор алгоритмов, свободно смоделированных на подобии человеческого мозга, которые предназначены для распознавания закономерностей. Они интерпретируют сенсорные данные через своего рода машинное восприятие, маркировку или кластеризацию исходных данных. Распознаваемые ими закономерности являются числовыми, содержатся в векторах, в которые должны быть переведены все реальные данные, будь то изображения, звук, текст или временные ряды. Нейронные сети помогают нам группировать и классифицировать. О них можно думать, как о кластеризации и классификации поверх данных, которые хранятся и которыми управляют. Они помогают группировать немаркированные данные в соответствии с сходством между входами примеров, и они классифицируют данные, когда у них есть маркированный набор данных для обучения. Соединения биологического нейрона моделируются в виде весов. Положительный вес отражает возбуждающую связь, отрицательные значения означают ингибирующие (тормозные) связи. Все входные данные модифицируются по весу и суммируются. Эта активность называется линейной комбинацией. Данная комбинация проходит через функцию активации, которая контролирует амплитуду выхода.

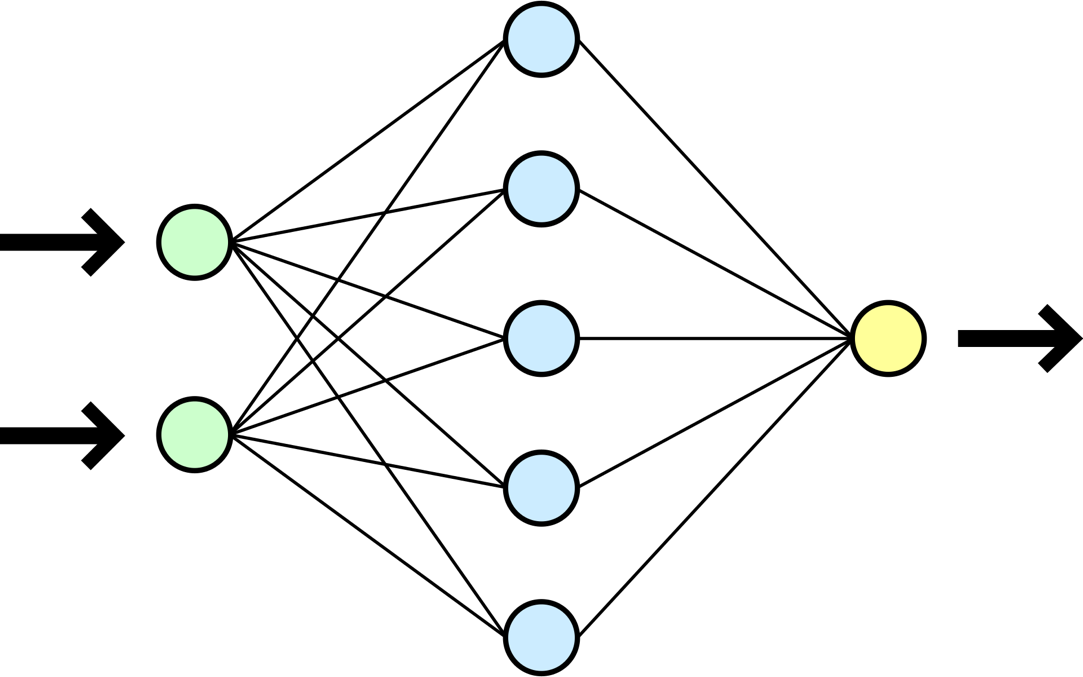


Рисунок 1 – Классический вид нейронной сети

**2.2. Сверточные нейронный сети**

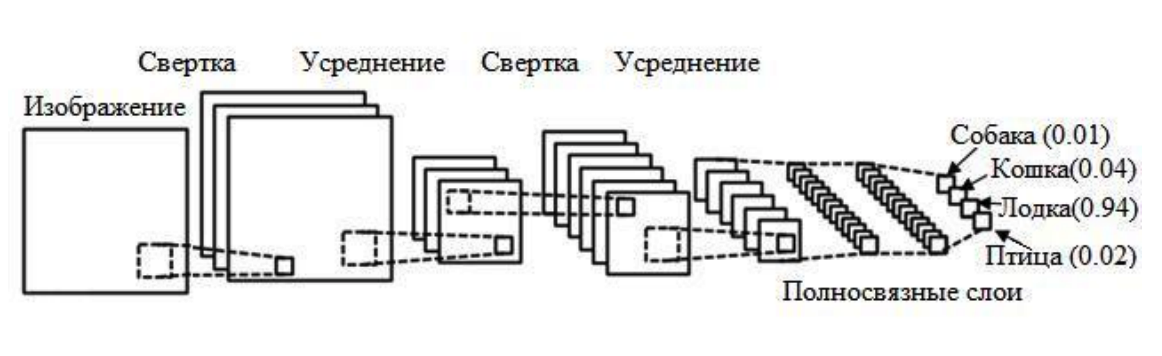
Основной смысл в сверточных нейронных сетях - сверточные слои изучают локальные шаблоны в случае с изображениями — шаблоны в небольших двумерных окнах во входных данных. Поэтому можно сказать, что сверточная нейронная сеть делает сильные и в основном правильные предположения о природе изображений, а именно о локальности пиксельных зависимостей и стационарности статистики. Следовательно, по сравнению со стандартными нейронными сетями с прямой связью, сверточная нейронная сеть имеет гораздо меньше соединений и параметров, что облегчает обучение. Сверточная нейронная сеть состоит из сверточных слоев, за которыми следуют полностью связанные слои (рис. 2). Сверточный слой состоит из фильтра свертки, за которым следуют фильтр объединения и функция активации. 

Рисунок 2 – Типичная архитектура для СНС

Сверточный фильтр имеет несколько фильтров с тем же размером окна, которые растекаются по изображению с заданным шагом. Объединение суммирует выходные данные соседних групп нейронов в одну ячейку – ядро. 23 Фильтр объединения имеет окно с определенным размером, которое перемещается по изображению с заданным шагом. Обычно, функцией объединения является функция максимума, которая выводит максимальное значение в ячейку, т.е. в ядро и используется в нашей модели

**2.3. Модель Inception**

Inception – архитектура моделей сверточных нейронных сетей, разработанная компанией Google. На текущий момент времени известно несколько версий модели с различными модификациями, которые позволяют увеличить скорость работы, а также уменьшить количество задаваемых параметров (чем меньше параметров, тем выше скорость обучения и функционирования). Так же стоит отметить существование модели Inception-ResNet. Данная модель была так же разработана компанией Google. Особенность этой модели заключается в наличии блоков ResNet в модели Inception. На рисунке 3 представлена архитектура Inception. Можно заметить, что она состоит из слоев и блоков (совокупности слоев). На рисунках 4 и 5 представлены блоки основного и вспомогательного выходов нейронной сети, где:

AveragePool – слой пулинга по среднему значению

Conv – сверточный слой

FC – полносвязный слой

SoftMax Activation – слой активации

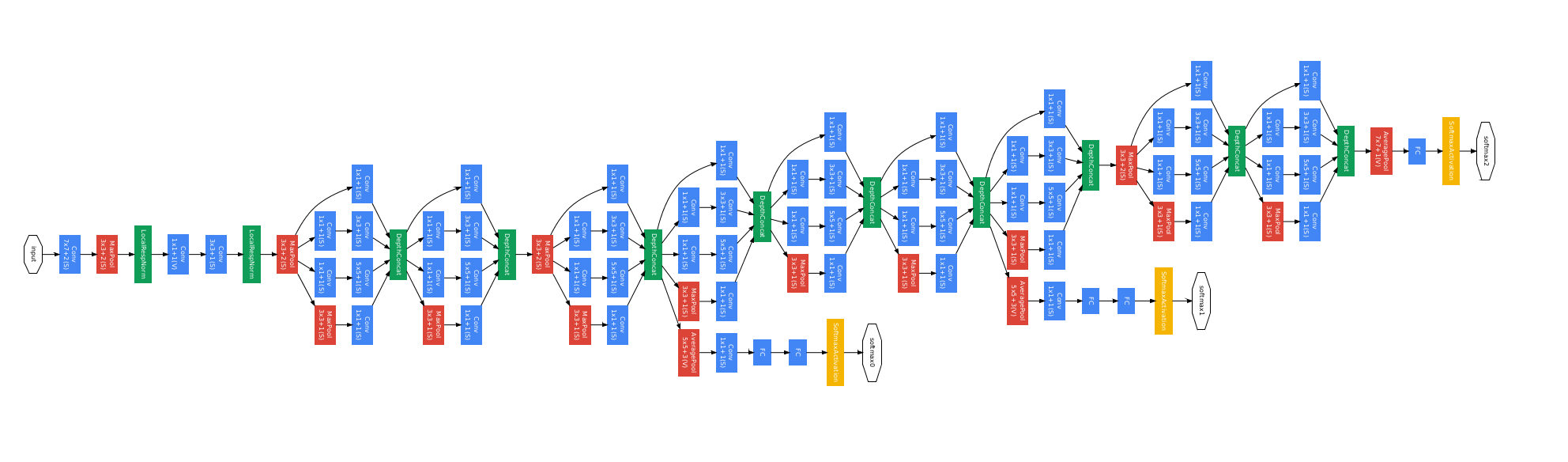


Рисунок 3 – Архитектура Inception

|  |  |
| --- | --- |
| Рисунок 5 – Вспомогательный выход | Рисунок 4 – Основной выход |

На рисунках 6 - 8 изображены блоки Inception. Эти блоки появились, благородя замене довольно «дорогой» операции свертки размером, например 5x5 на 2 более «дешевых» операций сверток с размером 3x3. Эта модификация позволяет уменьшить количество параметров, тем самым увеличив скорость и качество обучения сверточной нейронной сети. На рисунке 9 изображена структура сверточной нейронной сети Inception v2. Функцией активации на выходных слоях в данной сети служит логистическая регрессия (SoftMax).

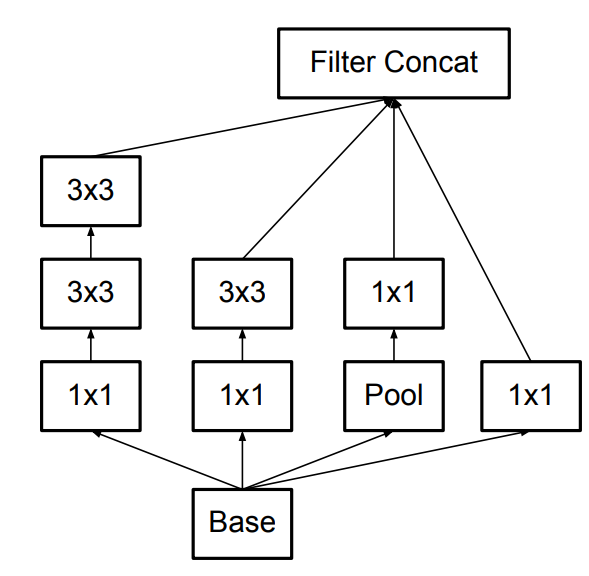


Рисунок 6 – Inception v2. Блок A

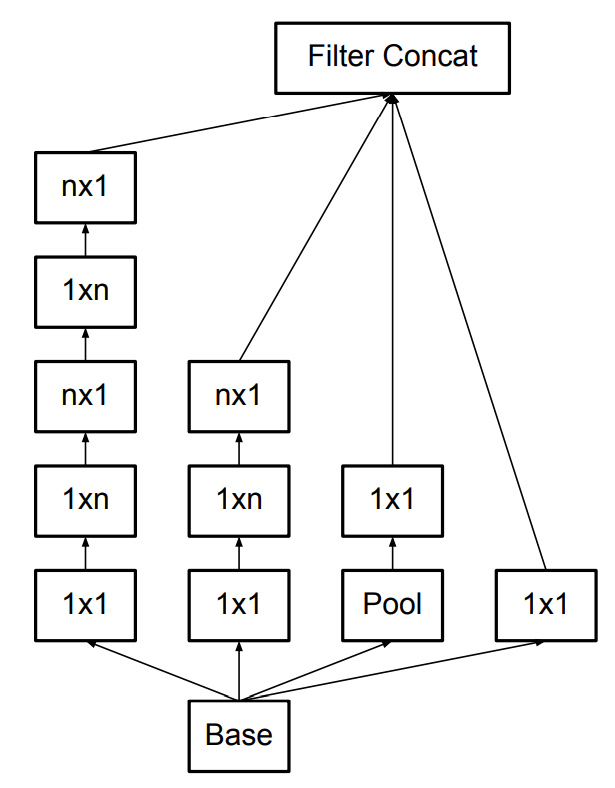


Рисунок 7 – Inception v2. Блок B

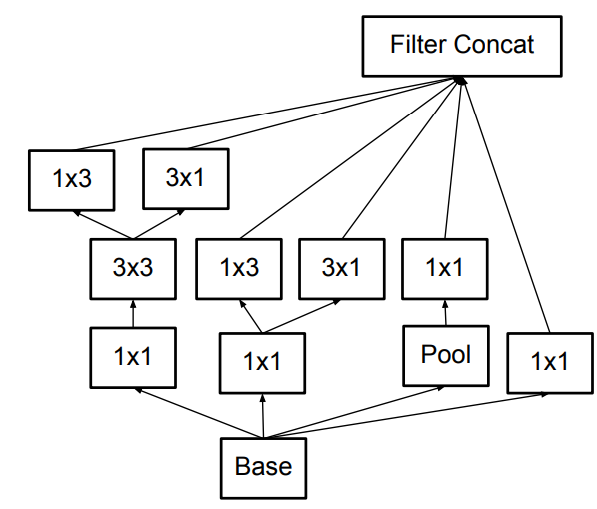


Рисунок 8 – Inception v2. Блок C

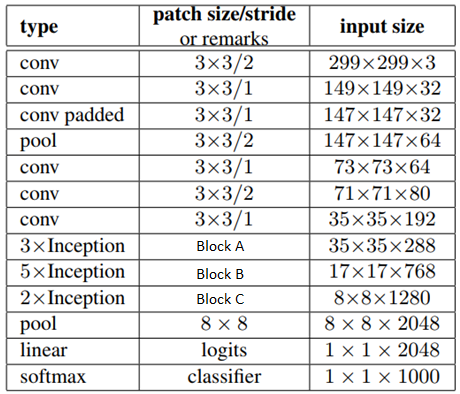


Рисунок 9– Структура нейронной сети Inception v2

В сверточных нейронных сетях большая часть работы заключается в выборе правильного слоя для применения. Среди наиболее распространенных вариантов: фильтр 1x1, фильтр 3x3, фильтр 5x5 или Max Pooling. Все, что нам нужно, это найти оптимальную локальную конструкцию и повторить ее оптимальное количество раз. Обычно фильтры накладываются друг на друга предполагая, что соотношение 3×3 и 5×5 должно увеличиваться по мере продвижения к более высоким слоям. Однако вычислительная стоимость такого решения значительно возрастает. По этой причине была создана сеть GoogLeNet, идея которой заключалась в том, что фильтры разных размеров, работают на одном уровне. Сеть, по сути, станет немного «шире», чем «глубже». Чтобы сделать его более дешевым, авторы ограничивают количество входных каналов, добавляя дополнительные свертки меньшего размера вместо одной свертки большего размера. Хотя добавление дополнительной операции может показаться нелогичным, но несколько фильтров 3x3 намного дешевле, чем один фильтр 5x5. Inception-v3 является третьей версией GoogLeNet. Переосмысливая начальную архитектуру, достигается вычислительная эффективность и меньшее количество параметров, а точнее 42-уровневой сети глубокого обучения, схожей с VGGNet. Благодаря 42 слоям достигается меньший коэффициент ошибок, что делает его первым занявшим второе место в классификации изображений в ILSVRC 2015.

**3. Реализация модели**

**3.1. Используемые программные средства**

Реализация модели в рамках данной работы производилась в среде разработки PyCharm , на языке Python v3.8. Список используемых библиотек:

* TFlearn - это модульная и прозрачная библиотека глубокого обучения, построенная на основе Tensorflow.
* OpenCV – загрузка и предобработка изображений;
* Matplotlib – построение различных графиков и диаграмм;
* NumPy – библиотека, необходимая для работы с векторами и массивами.

Также для обучения нейронной сети использовались данные, взятые с платформы Kaggle.

**3.2. Разработанные классы**

Для достижения заданной цели работы был написан класс main, содержащий в себе всю логику работы нашей модели. В классе реализовано несколько методов:

* create\_train\_data – метод, использующий возможности библиотеки OpenCV, который обрабатывает массив исходных изображений и приводит их к квадратному виду размера 50x50;
* process\_test\_data – метод, которые проводит те же операции с изображениями, что и методы create\_train\_data, только для тестовых файлов;
* оставшийся блок кода, который отвечает за тренировку нейронной сети, получения модели из обработанного DataSet, а также вывод результатов на главное окно программы.

Построение сверточной нейронной сети происходит с входного слоя:

convnet = input\_data(shape=[None, IMG\_SIZE, IMG\_SIZE, 1], name='input')

Далее у нас имеется 6 слоев свертки и объединения:

convnet = conv\_2d(convnet, 32, 2, activation='relu')  
convnet = max\_pool\_2d(convnet, 2)  
  
convnet = conv\_2d(convnet, 64, 2, activation='relu')  
convnet = max\_pool\_2d(convnet, 2)  
  
convnet = conv\_2d(convnet, 32, 2, activation='relu')  
convnet = max\_pool\_2d(convnet, 2)  
  
convnet = conv\_2d(convnet, 64, 2, activation='relu')  
convnet = max\_pool\_2d(convnet, 2)  
  
convnet = conv\_2d(convnet, 32, 2, activation='relu')  
convnet = max\_pool\_2d(convnet, 2)  
  
convnet = conv\_2d(convnet, 64, 2, activation='relu')  
convnet = max\_pool\_2d(convnet, 2)

Затем добавляем полностью связанный слой:

convnet = fully\_connected(convnet, 1024, activation='relu')  
convnet = dropout(convnet, 0.8)

Также необходимо создать выходной слой:

convnet = fully\_connected(convnet, 2, activation='softmax')  
convnet = regression(convnet, optimizer='adam', learning\_rate=LR, loss='categorical\_crossentropy', name='targets')

И наконец библиотека TFlearn предоставляет оболочку модели DNN, с помощью которой мы создаем глубокую нейронную сеть на основе заданной архитектуры. Мы будем запускать ее в течении 5 эпох (т.е. сеть будет просматривать все данные 5 раз).

model = tflearn.DNN(convnet, tensorboard\_dir='log')

Если посмотреть вывод в консоли (рис. 10), можно заметить, что наша модель завершила тренировку с точность порядка 93%, что означает, что она может предсказывать правильный результат для 93% от общего числа кошек и собак.

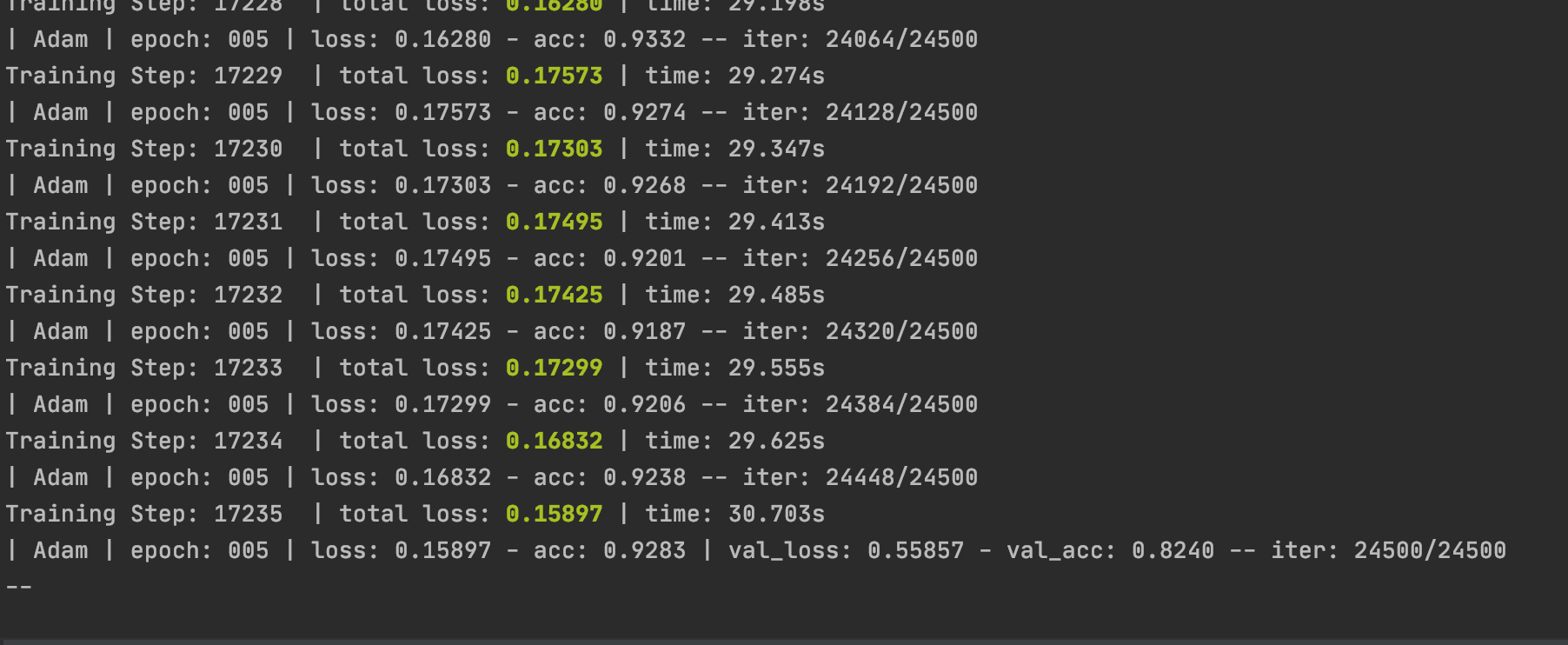


Рисунок 9– Процент точности обучения

**3.3. Тестирование модели**

Для тестирования модели был использован готовый тестовый датасет с платформы Kaggle, содержащий в себе порядка 25 000 картинок. Для наглядного отображения результатов была необходимость в использовании библиотеки matplotlib, тестирование модели изображено на рисунке 10. Для того, чтобы выходные результаты были четче и яснее, использовалось 25 случайных изображений кошек и собак, а также красная надпись над рисунком, показывающая изображенное животное по мнению алгоритма.

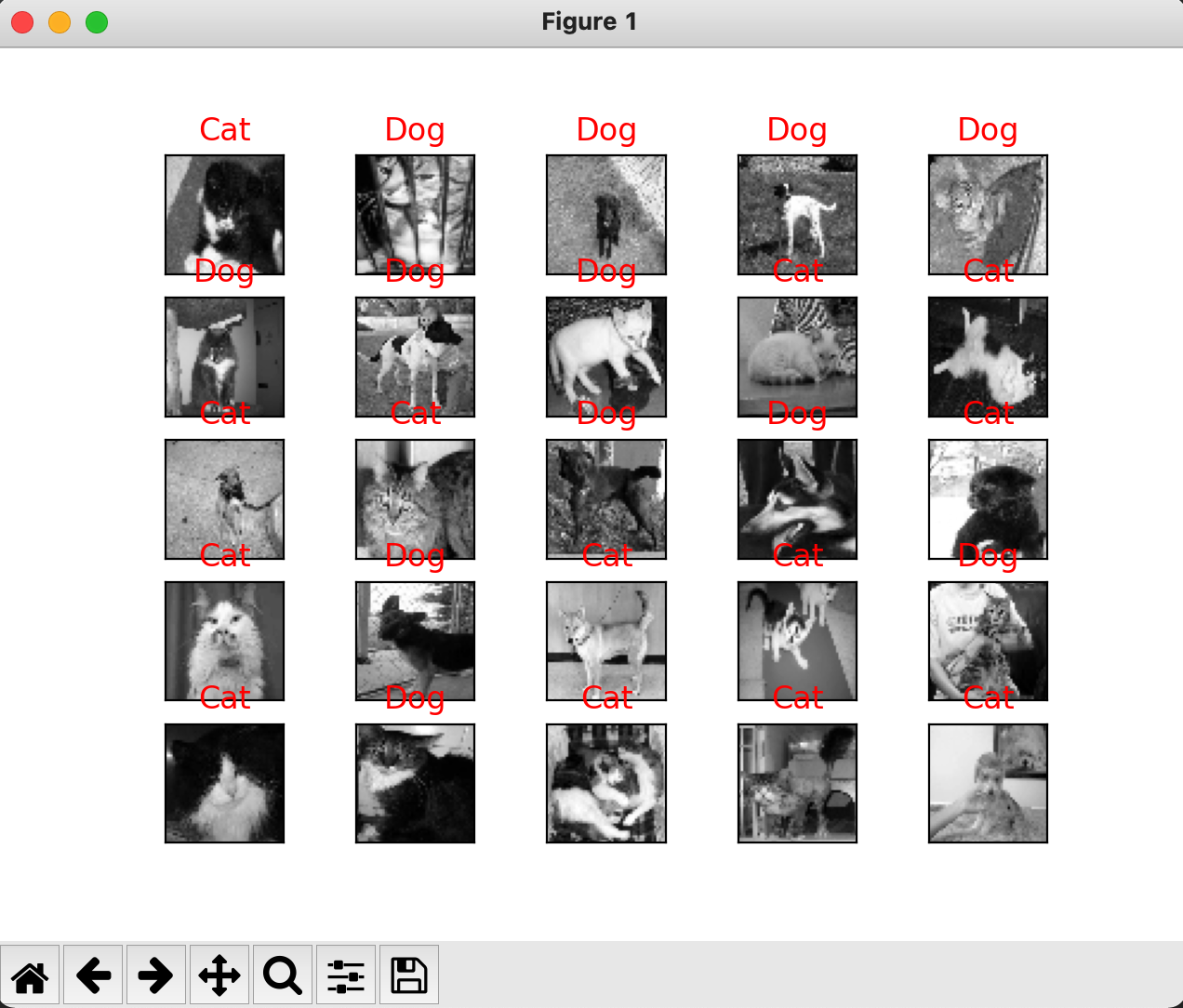


Рисунок 10 – Изображения, использованные для тестирования

Все 25000 тестовых картинок были обработаны алгоритмом и результаты вынесены в текстовый файл. Для примера часть данного файла представлена на рисунке 11

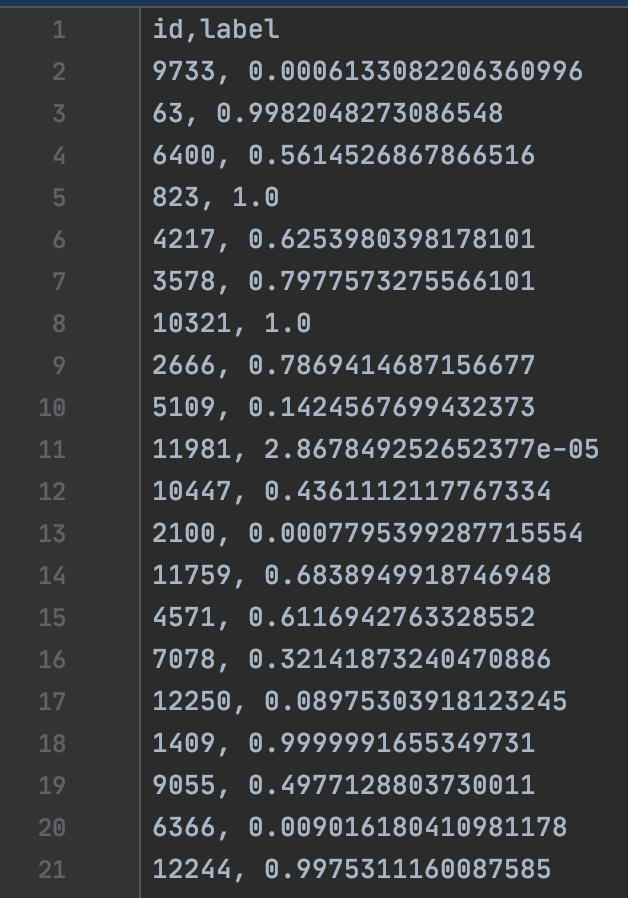


Рисунок 11 – Полученные гистограммы распределений классов для данных изображений

В колонке label отражается уверенность алгоритма в определенном классе. Чем ближе данное значение к 1, тем увереннее наша модель определяет, что это собака.

Также данный .csv файл можно загрузить на платформу Kaggle и проверить самого себя, насколько получившаяся модель работает успешно по сравнению с другими участниками, которые решали похожую задачу. Получившийся результат изображен на (рис. 12).

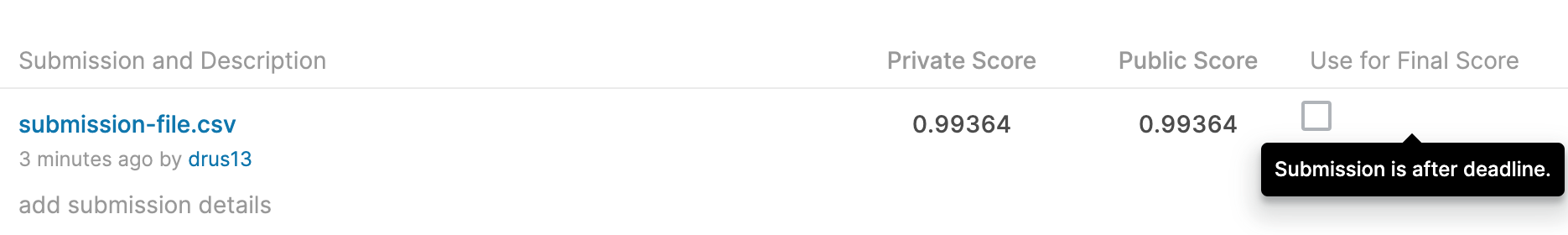


Рисунок 12 – Полученный результат по мнению платформы Kaggle

**4. Заключение**

Подводя итоги, можно сделать вывод, что на сегодняшний момент существуют разные архитектуры сверточных нейронных сетей, способные решать задачи обнаружения объектов в реальном времени. Эта возможность появилась благодаря развитию машинного обучения, в частности очень глубоких сетей, и аппаратной части для обучения таких ресурсозатратных нейронных сетей.

В данной курсовой работе была изучена и создана модель нейронной сети Inception с возможностью классифицировать несколько видов объектов на изображении.

Проведено тестирование полученной модели и достигнута точность в 93%

**5. Список использованных источников**

* <https://habr.com/ru/post/302242/>
* <https://habr.com/ru/post/321834/>
* <https://cloud.google.com/tpu/docs/inception-v3-advanced>

**Приложение.**

import cv2  
import numpy as np  
import os  
from random import shuffle  
from tqdm import tqdm  
  
TRAIN\_DIR = '/Users/""/Downloads/dogs-vs-cats/train'  
TEST\_DIR = '/Users/""/Downloads/dogs-vs-cats/test1'  
IMG\_SIZE = 50  
LR = 1e-3  
  
MODEL\_NAME = 'dogsvscats-{}-{}.model'.format(LR, '6conv-basic-video')  
  
  
def label\_img(img):  
 word\_label = img.split('.')[-3]  
 if word\_label == 'cat':  
 return [1, 0]  
 elif word\_label == 'dog':  
 return [0, 1]  
  
  
def create\_train\_data():  
 training\_data = []  
 for img in tqdm(os.listdir(TRAIN\_DIR)):  
 label = label\_img(img)  
 path = os.path.join(TRAIN\_DIR, img)  
 img = cv2.resize(cv2.imread(path, cv2.IMREAD\_GRAYSCALE), (IMG\_SIZE, IMG\_SIZE))  
 training\_data.append([np.array(img), np.array(label)])  
 shuffle(training\_data)  
 np.save('train\_data.npy', training\_data)  
 return training\_data  
  
  
#train\_data = create\_train\_data()  
train\_data = np.load('train\_data.npy', '', True, True, 'ASCII')  
  
  
def process\_test\_data():  
 testing\_data = []  
 for img in tqdm(os.listdir(TEST\_DIR)):  
 path = os.path.join(TEST\_DIR, img)  
 img\_num = img.split('.')[0]  
 img = cv2.resize(cv2.imread(path, cv2.IMREAD\_GRAYSCALE), (IMG\_SIZE, IMG\_SIZE))  
 testing\_data.append([np.array(img), img\_num])  
  
 np.save('test\_data.npy', testing\_data)  
 return testing\_data  
  
  
import tflearn  
from tflearn.layers.conv import conv\_2d, max\_pool\_2d  
from tflearn.layers.core import input\_data, dropout, fully\_connected  
from tflearn.layers.estimator import regression  
  
from tensorflow.python.framework import ops  
  
ops.get\_default\_graph()  
  
convnet = input\_data(shape=[None, IMG\_SIZE, IMG\_SIZE, 1], name='input')  
  
convnet = conv\_2d(convnet, 32, 2, activation='relu')  
convnet = max\_pool\_2d(convnet, 2)  
  
convnet = conv\_2d(convnet, 64, 2, activation='relu')  
convnet = max\_pool\_2d(convnet, 2)  
  
convnet = conv\_2d(convnet, 32, 2, activation='relu')  
convnet = max\_pool\_2d(convnet, 2)  
  
convnet = conv\_2d(convnet, 64, 2, activation='relu')  
convnet = max\_pool\_2d(convnet, 2)  
  
convnet = conv\_2d(convnet, 32, 2, activation='relu')  
convnet = max\_pool\_2d(convnet, 2)  
  
convnet = conv\_2d(convnet, 64, 2, activation='relu')  
convnet = max\_pool\_2d(convnet, 2)  
  
convnet = fully\_connected(convnet, 1024, activation='relu')  
convnet = dropout(convnet, 0.8)  
  
convnet = fully\_connected(convnet, 2, activation='softmax')  
convnet = regression(convnet, optimizer='adam', learning\_rate=LR, loss='categorical\_crossentropy', name='targets')  
  
model = tflearn.DNN(convnet, tensorboard\_dir='log')  
  
if os.path.exists('{}.meta'.format(MODEL\_NAME)):  
 model.load(MODEL\_NAME)  
 print('model loaded')  
  
train = train\_data[:-500]  
test = train\_data[-500:]  
  
X = np.array([i[0] for i in train]).reshape(-1, IMG\_SIZE, IMG\_SIZE, 1)  
Y = [i[1] for i in train]  
  
test\_x = np.array([i[0] for i in test]).reshape(-1, IMG\_SIZE, IMG\_SIZE, 1)  
test\_y = [i[1] for i in test]  
  
model.fit({'input': X}, {'targets': Y}, n\_epoch=5, validation\_set=({'input': test\_x}, {'targets': test\_y}),  
 snapshot\_step=500, show\_metric=True, run\_id=MODEL\_NAME)  
  
model.save(MODEL\_NAME)  
  
import matplotlib.pyplot as plt  
  
  
#test\_data = process\_test\_data()  
test\_data = np.load('test\_data.npy', '', True, True, 'ASCII')  
  
fig = plt.figure()  
  
for num, data in enumerate(test\_data[:25]):  
 # cat[1,0]  
 # dog[0,1]  
  
 img\_num = data[1]  
 img\_data = data[0]  
  
 y = fig.add\_subplot(5,5,num+1)  
 orig = img\_data  
 data = img\_data.reshape(IMG\_SIZE, IMG\_SIZE, 1)  
  
 model\_out = model.predict([data])[0]  
  
 if np.argmax(model\_out) == 1: str\_label = 'Dog'  
 else: str\_label = 'Cat'  
  
 y.imshow(orig, cmap= 'gray')  
 title\_obj = plt.title(str\_label)  
 plt.setp(title\_obj, color = 'r')  
 y.axes.get\_xaxis().set\_visible(False)  
 y.axes.get\_yaxis().set\_visible(False)  
plt.show()  
  
with open('submission-file.csv', 'w') as f:  
 f.write('id,label\n')  
  
with open('submission-file.csv', 'a') as f:  
 for data in tqdm(test\_data):  
 img\_num = data[1]  
 img\_data = data[0]  
 orig = img\_data  
 data = img\_data.reshape(IMG\_SIZE, IMG\_SIZE, 1)  
 model\_out = model.predict([data])[0]  
 f.write('{}, {}\n'.format(img\_num, model\_out[1]))