**ADN-Automatic Driving Network**

**A Project Report**

Submitted in partial fulfilment of the

Requirements for the award of the Degree of

**BACHELOR OF SCIENCE (COMPUTER SCIENCE)**

**By**

**Amritpal Singh Randhawa**

**Roll Number - 408**

**Under the esteemed guidance of**

**Prof. Randeep Singh Ghai**

**Assistant Professor**



**DEPARTMENT OF COMPUTER SCIENCE**

**GURU NANAK KHALSA COLLEGE**

**OF**

**ARTS, SCIENCE & COMMERCE**

***(Autonomous)***

**MATUNGA, MUMBAI – 400 019**

**MAHARASHTRA**

**AY 2019 – 2020**

**APPROVAL OF PROJECT PROPOSAL**



**PNR No: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Roll No: \_\_\_\_\_\_\_\_**

**1. Name of the Student**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**2. Title of the Project**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**3. Name of the Project Guide**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**4. Teaching experience of the Project Guide - \_\_\_\_\_\_\_\_\_\_\_**

**5. Is this your first submission? \_\_\_\_\_\_\_\_\_\_\_**

**Signature of the Student Signature of the Guide**

**Date: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Date: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Signature of the Head of Dept. Signature of the Examiner**

**Date: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Date: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**GURU NANAK KHALSA COLLEGE**

**OF**

**ARTS, SCIENCE & COMMERCE**

***(Autonomous)***

**MATUNGA, MUMBAI, MAHARASHTRA – 400 019**

**DEPARTMENT OF COMPUTER SCIENCE**



**CERTIFICATE**

This is to certify that the project entitled, "**AND-Automated driving network** ", is bonafide work of **Amritpal Singh Randhawa** bearing Seat No: **408** submitted in partial fulfilment of the requirements for the award of degree of BACHELOR OF SCIENCE (COMPUTER SCIENCE) from University of Mumbai.

**Internal Guide Coordinator**

**External Examiner**

**Date: College Seal**

**Abstract**

This is a convolutional neural network which is specifically designed

for driving in games on its own. The game is not a restriction as any

game that has a driving logic is compatible with this code. For this

project a neural network is trained on NFS Most Wanted 2012 with a

custom game data set to let the code know the game and learn to

drive in it.

This network is optimized to run on low-end devices of computer spectrum. InceptionV3 model is used to train this specific network but a variety of model are written to choose from for training purpose.

**Acknowledgement**

This major project would not have been possible without the valuable assistance of many people to whom we are indebted, in particular, our project coordinator RANDEEP SINGH GHAI of GN KHALSA College. We would also like to thank “CS Department”, **GN KHALSA MATUNGA** for providing us with the necessary components for our project. Our thanks also go to all the teacher’s CS Department who helped us in many difficult situations regarding the project and provided with the necessary advice. A special word of thanks is to our class mates and our families for providing us the moral support.

**DECLARATION**

I hereby declare that the project entitled, “**AND-Automatic driving network**” done at **Guru Nanak Khalsa College**, has not been in any case duplicated to submit to any other university for the award of any degree. To the best of my knowledge other than me, no one has submitted to any other university.

The project is done in partial fulfilment of the requirements for the award of degree of **BACHELOR OF SCIENCE (COMPUTER SCIENCE)** to be submitted as final semester project as part of our curriculum.

**Name of the Student**

**TABLE OF CONTENTS**

**CHAPTER 1: INTRODUCTION**

* 1. **Background**
  2. **Proposed system**
  3. **Scope**

**CHAPTER 2: SURVEY OF TECHNOLOGIES**

**2.1 Problem**

**2.2 Neural Networks in Self-Playing AI**

**CHAPTER 3: REQUIREMENTS AND ANALYSIS**

**3.1 Feasibility Study**

**3.2 Software and Hardware Requirements**

**3.3 Planning and Scheduling**

**3.3.1 GANTT Chart**

**3.3.2 WBS Chart**

**3.4 Conceptual model**

**CHAPTER 4:CODE**

**CHAPTER 1: INTRODUCTION**

This is a convolutional neural network which is specifically designed for driving in games on its own. The game is not a restriction as any game that has a driving logic is compatible with this code. For this project a neural network is trained on NFS Most Wanted 2012 with a custom game data set to let the code know the game and learn to drive in it.

Neural Networks (NN) technology is one of the most used approaches in modern Artificial Intelligence (AI). It has been applied successfully to solving such problems as forecasting, adaptive control, recognition classification, and many others.

An artificial NN is a simple model of a biological brain. It consists of elements called neurons. An artificial neuron is just a simple math model of biological neuron. Because an artificial NN is modelled after the biological brain, it has similar conceptual properties such as the capability of learning.

Convolutional Neural Networks (CNN) and Deep Learning (DL) are related branches of NN computing that have been developed in recent years. CNN is a neural network with a special structure that was designed as a model of a human vision system (HVS). Thus, CNNs are most suitable for solving problems of computer vision, such as object recognition and classification of images and video data. They have also been used successfully for speech recognition and text translation.

The increasing popularity of DL technology has influenced the development of many new CNN programming frameworks. The most popular frameworks are Caffe, TensorFlow, Theano, Torch and Keras.

CUDA (Compute Unified Device Architecture) is a parallel computing platform and API model created by Nvidia. It allows software developers and software engineers to use a CUDA-enabled graphics processing unit (GPU) for general purpose processing – an approach termed GPGPU (General-Purpose computing on Graphics Processing Units). The CUDA platform is a software layer that gives direct access to the GPU's virtual instruction set and parallel computational elements, for the execution of compute kernels.

The CUDA platform is designed to work with programming languages such as C, C++, and Fortran. This accessibility makes it easier for specialists in parallel programming to use GPU resources, in contrast to prior APIs like Direct3D and OpenGl, which required advanced skills in graphics programming. CUDA-powered GPUs also support programming frameworks such as OpenACC and OpenCL; and HIP by compiling such code to CUDA.

**1.2 Proposed System:**

This Neural network is not hardcoded for only one game it can be implemented in any game where driving is part of gameplay. This system is notched down to work on a low-end device.

A variety of models are written that can be used to train the network accordingly. This is compatible with any device which can handle CUDA i.e. the graphic cards that support CUDA.

**1.3 Scope:**

This project focusses on creating a simple Neural Network for autonomous

Racing car. This requires active collection of data from environment and to make decision according to the data received. To achieve these high-end devices are used but this network should be able to run on a low-end device for making majority of people who aspire to develop NN work with something. Main focus of this project is to optimize big networks without compromising with accuracy that they work with.

**CHAPTER 2: SURVEY OF TECHNOLOGIES**

**2.1 Problem:**

Computer games bots are pieces of software that are able to play computer games autonomously. Researchers have applied many different techniques to the design of bots, and one such modern technique is the development and training of neural networks (and other learning machines) to play single-player computer games, as training is made easy. Neural networks require many thousands upon thousands of data points, which also assumes the behaviour being replicated must be able to be encoded (for example, for a feed-forward neural network with a single layer of hidden neurons, the behaviour must be able to be represented as a continuous bounded function. Fortunately, virtually every aspect of a computer game is already encoded and accessible, and often data can be very easily generated to provide a large enough training set.

However, most neural networks of this nature are done for single-player computer games, where all other agents are programmed. Unsupervised neuro-evolutionary computing is a popular method in this instance, where the goal is unchanging, but the solution is unknown. Neuro-evolutionary computing is a kind of reinforcement learning algorithm first proposed by Kenneth O. Stanley, where a generation of random neural networks is generated, and is then tested and rated against some reward function. Poor models are killed off, whilst fitter models are” bred” to make a new generation. This has been done for Super Mario World as well as for a host of games for the Atari 2600, to name just a few examples.

**2.2** **Neural Networks in Self-Playing AI**

With regards to the implementation of the neural network, it was decided early-on that the project would likely make use of a particular model of neural network known as a recurrent neural network. As opposed to the traditional feed-forward neural network, a recurrent neural network not only processes inputs from one layer to the next (i.e. feeding the in-formation forward), but information is also retained and fed back to other nodes within the feature space. This allows for information to be retained between frames, creating the possibility for patterns to be recognised within gameplay, because it allows time to be represented. Especially promising is the model known as the” long short-term memory” network, a form of recurrent network that has a very large memory capacity compared to a normal RNN.

LSTM networks are comprised of LSTM cells. Each unit is typically comprised of an input gate, an output gate, a cell that stores the state, and a forget gate. Together, these structures decide, each time step, what information will be forgotten, what new information will be stored, then use this and the currently stored information to decide what to output. This was explained in depth by Colah.

An RNN with external memory was used to train multiple agents to play Quake III Arena , but using an LSTM solves the same problem as the utilisation of external memory - the problem of the limitation of memorisation due to the gradient vanishing and exploding problem, which is due to ”the temporal evolution of the backpropagated error exponentially depending on the size of the weights” where back-propagation is a common and efficient method of training networks. A Long Short-Term Memory network was also used to train Mari Flow, a self-learning Super Mario Kart AI.

The first segment of this problem, however, was the matter of feature pre-processing -obtaining live information regarding the state of the game from frame to frame. Such information was not accessible using an API, so the only way to approach this problem was to utilise computer vision solutions. Research on OpenCV, the primary and most well-documented computer vision module for Python, was conducted primarily via its own documentation. In addition, the computer vision section of a course was followed, run by Harrison Kinsley, AKA "Sentdex" .

**CHAPTER 3: REQUIREMENTS & ANALYSIS**

**3.1 FEASIBILITY STUDY**

This phase of the analysis is based on the feasibility study of the

proposed system. Any system is feasible when given infinite time

and resources. The problem is whether the proposed system

achieves the goal by using available resources and pre-specified

time.

For this to start any project, we have to study the system so that

the requirements and goals should be matched. This feasibility

study involves:

* Technical feasibility
* Economic feasibility
* Operational feasibility

**Technical feasibility:**

The proposed system needs the data in an efficient manner i.e.,

it needs protection of data and authorized access to the data.

This is done when we use the software, which supports console-oriented methodology and database methodology.

Therefore, technical feasibility determines whether the organization has the technology and skills necessary to carry out the project and how this is obtained. The system is technically feasible on the following grounds**:**

* All necessary technology is available those that are required to develop the system.
* The existing resource is capable and can hold all the necessary data in an efficient way.
* The system is too flexible and can expanded further.
* The system can give guarantee of accuracy, ease of use, reliability and data-security.
* The system can give instant responses**.**

**The following are the activities that are undertaken during this study:**

* Development Risks
* Determining whether the system can be designed so that the necessary function and performance is achieved within the constraints uncovered during the analysis**.**

**Resource Availability:**

It specifies whether the hardware and software resources necessary to develop the system is available. Connecting to Oracle is very suitable to accomplish the task.

The organization has bought the required hardware and software, these efforts from the organization made the proposed system technically feasible.

**Technology:**

Whether the relevant technology progressed to such a state that it could support the system. So, we can conclude that the project is technically feasible.

**Economic Feasibility:**

It determines whether the project goal can be within the resource limits allocated to it.

It must determine whether it is worthwhile to process with the project or whether the benefit obtained from the new system is not worth the costs. After conducting the cost benefit analysis, it reveals that the objectives of the proposed system can be achieved within the allocated resources. The proposed system requires no extra man-power.

Also, the cash investment to implement the proposed system can be easily recovered.

**Operational Feasibility:**

This determines if the proposed system has satisfied user objectives and can be fit into the current system operation. The present system is operationally feasible on the following grounds:

* The method of processing and presentation are completely accepted to the clients since they can meet all the requirements.
* The clients have been involved in the planning and development of the system.
* The proposed system will not cause any problem under any circumstances.
* The proposed system will certainly satisfy the user requirements and will also enhance their capabilities. It can be best fit into current operations. Also, the maintenance of the system is very easy and requires minimal persons. Therefore, the system is operationally feasible.

**3.2 Software and Hardware Requirements**

**3.2.1 Hardware Requirements:**

Processor: Intel icore i5 8th Gen

System bus: 64 bits

RAM: 8 GB of RAM

Hard Drive: 1TB

**3.2.2 Software Requirements:**

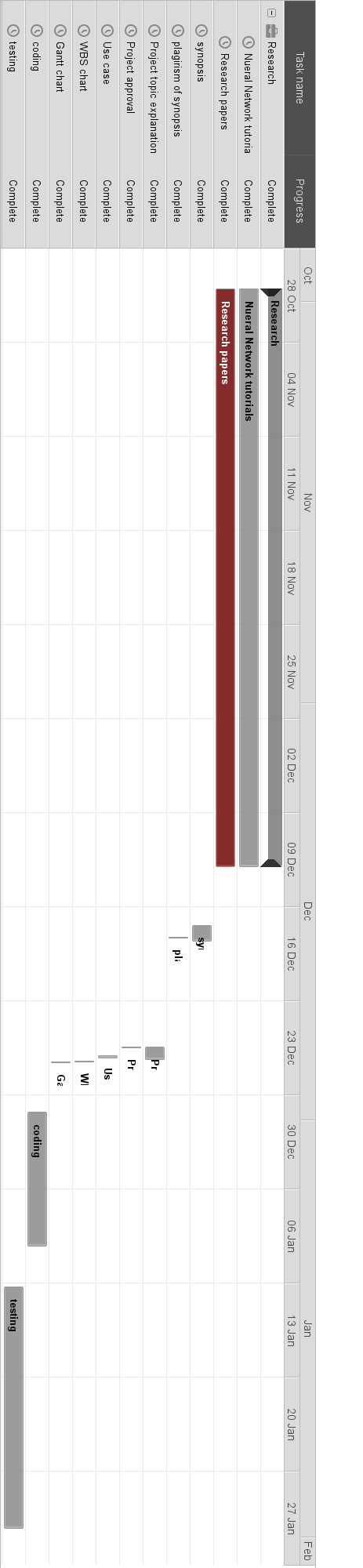
OS: MS Windows 10

It also needs:

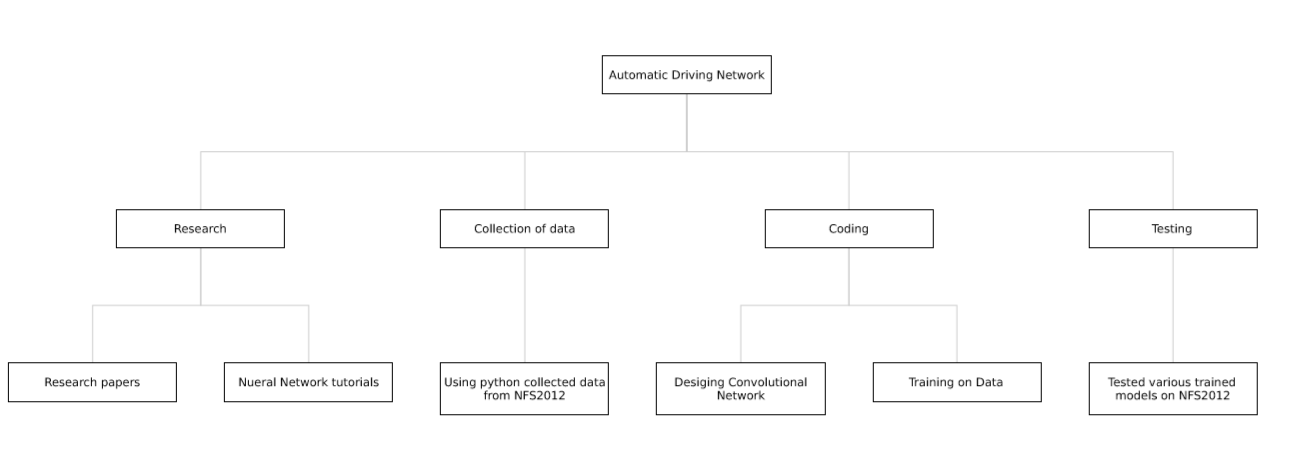
* Python 3.6(64 bit)
* Google-auth 1.8.2
* Google-auth-oauthlib 0.4.1
* Keras-Applications 1.0.8
* Kears-Preprocessing 1.1.0
* Numpy 1.16.1
* OpenCV-python 4.1.2.30
* Pandas 0.25.3
* Pip 19.3.1
* Pypiwin32 223
* Pywin32 227
* Tensorboard 1.8.0
* Tensorflow 1.5.0
* Tensorflow-estimator 1.15.1
* Tensorflow-gpu 1.5.0
* Tensorflow-tensorboard 1.5.1
* Tflearn 0.3.2
* Torch 1.3.1
* Torchvision 0.4.2
* CUDA 9

**3.3 Planning and Scheduling**

**3.3.1** **GANTT Chart**--

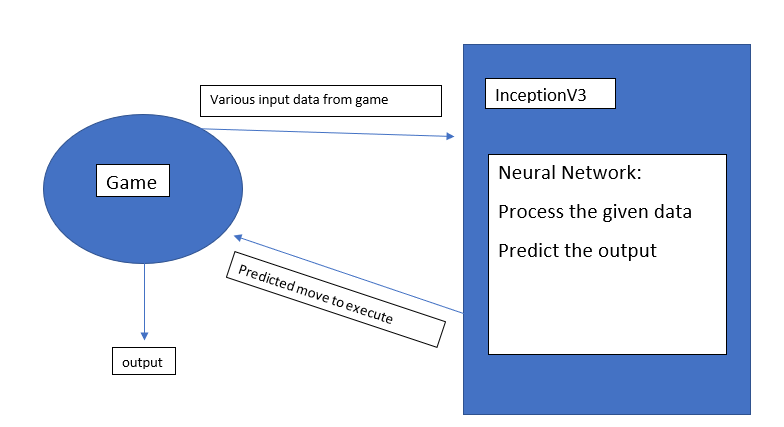
****

**3.3.2 WBS Chart-**

****

**3.4 Conceptual Models-**

USE CASE DIAGRAM-



**Chapter 4 CODE:**

1.Collecting data:

import os, time, cv2

import numpy as np

from grabscreen import grab\_screen

from getkeys import key\_check

key\_map = {

'W': [1, 0, 0, 0, 0, 0, 0, 0, 0],

'S': [0, 1, 0, 0, 0, 0, 0, 0, 0],

'A': [0, 0, 1, 0, 0, 0, 0, 0, 0],

'D': [0, 0, 0, 1, 0, 0, 0, 0, 0],

'WS': [0, 0, 0, 0, 1, 0, 0, 0, 0],

'WD': [0, 0, 0, 0, 0, 1, 0, 0, 0],

'SA': [0, 0, 0, 0, 0, 0, 1, 0, 0],

'SD': [0, 0, 0, 0, 0, 0, 0, 1, 0],

'NK': [0, 0, 0, 0, 0, 0, 0, 0, 1],

'default': [0, 0, 0, 0, 0, 0, 0, 0, 0],

}

starting\_value = 0

while True:

file\_name = 'training\_data-{0}.npy'.format(starting\_value)

if os.path.isfile(file\_name):

print('File exists, moving along', starting\_value)

starting\_value += 1

else:

print('File does not exist, starting fresh!', starting\_value)

break

def keys\_to\_output(keys):

'''

Convert keys to a ...multi-hot... array

0 1 2 3 4 5 6 7 8

[W, S, A, D, WA, WD, SA, SD, NOKEY] boolean values.

'''

if ''.join(keys) in key\_map:

return key\_map[''.join(keys)]

return key\_map['default']

def main(file\_name, starting\_value):

training\_data = []

for i in list(range(4))[::-1]:

print(i + 1)

time.sleep(1)

last\_time = time.time()

paused = False

print('STARTING!!!')

while(True):

if not paused:

screen = grab\_screen(region = (0, 40, 800, 600))

last\_time = time.time()

screen = cv2.resize(screen, (480, 270))

screen = cv2.cvtColor(screen, cv2.COLOR\_BGR2RGB)

keys = key\_check()

output = keys\_to\_output(keys)

training\_data.append([screen,output])

last\_time = time.time()

if len(training\_data) % 100 == 0:

print(len(training\_data))

if len(training\_data) == 2000:

np.save(file\_name, training\_data)

print('SAVED')

training\_data = []

starting\_value += 1

file\_name = 'training\_data-{0}.npy'.format(starting\_value)

keys = key\_check()

if 'T' in keys:

if paused:

paused = False

print('Unpaused!')

time.sleep(1)

else:

print('Pausing!')

paused = True

time.sleep(1)

if \_\_name\_\_ == "\_\_main\_\_":

main(file\_name, starting\_value)

2.Training data:

import os, cv2

import pandas as pd

import numpy as np

from grabscreen import grab\_screen

from tqdm import tqdm

from collections import deque

from models import inception\_v3 as googlenet

from random import shuffle

FILE\_I\_END =2000

WIDTH = 480

HEIGHT = 270

LR = 1e-3

EPOCHS = 3

MODEL\_NAME = ''

PREV\_MODEL = ''

LOAD\_MODEL = False

model = googlenet(WIDTH, HEIGHT, 3, LR, output=9, model\_name=MODEL\_NAME)

if LOAD\_MODEL:

model.load(PREV\_MODEL)

print('We have loaded a previous model!!!!')

for e in range(EPOCHS):

data\_order = [i for i in range(1, FILE\_I\_END + 1)]

for count, i in enumerate(data\_order):

try:

file\_name = 'training\_data-{0}.npy'.format(i)

train\_data = np.load(file\_name)

print('training\_data-{0}.npy'.format(i), len(train\_data))

train = train\_data[:-50]

test = train\_data[-50:]

X = np.array([i[0] for i in train]).reshape(-1, WIDTH, HEIGHT, 3)

Y = [i[1] for i in train]

test\_x = np.array([i[0] for i in test]).reshape(-1, WIDTH, HEIGHT, 3)

test\_y = [i[1] for i in test]

model.fit({'input': X}, {'targets': Y}, n\_epoch = 1, validation\_set = ({'input': test\_x}, {'targets': test\_y}),

snapshot\_step = 2500, show\_metric = True, run\_id = MODEL\_NAME)

if count % 30 == 0:

print('SAVING MODEL!')

model.save(MODEL\_NAME)

except Exception as e:

print(e)

3.Testing data:

import os, cv2

import pandas as pd

import numpy as np

from grabscreen import grab\_screen

from tqdm import tqdm

from collections import deque

from models import inception\_v3 as googlenet

from random import shuffle

FILE\_I\_END =2000

WIDTH = 480

HEIGHT = 270

LR = 1e-3

EPOCHS = 3

MODEL\_NAME = ''

PREV\_MODEL = ''

LOAD\_MODEL = False

model = googlenet(WIDTH, HEIGHT, 3, LR, output=9, model\_name=MODEL\_NAME)

if LOAD\_MODEL:

model.load(PREV\_MODEL)

print('We have loaded a previous model!!!!')

for e in range(EPOCHS):

data\_order = [i for i in range(1, FILE\_I\_END + 1)]

for count, i in enumerate(data\_order):

try:

file\_name = 'training\_data-{0}.npy'.format(i)

train\_data = np.load(file\_name)

print('training\_data-{0}.npy'.format(i), len(train\_data))

train = train\_data[:-50]

test = train\_data[-50:]

X = np.array([i[0] for i in train]).reshape(-1, WIDTH, HEIGHT, 3)

Y = [i[1] for i in train]

test\_x = np.array([i[0] for i in test]).reshape(-1, WIDTH, HEIGHT, 3)

test\_y = [i[1] for i in test]

model.fit({'input': X}, {'targets': Y}, n\_epoch = 1, validation\_set = ({'input': test\_x}, {'targets': test\_y}),

snapshot\_step = 2500, show\_metric = True, run\_id = MODEL\_NAME)

if count % 30 == 0:

print('SAVING MODEL!')

model.save(MODEL\_NAME)

except Exception as e:

print(e)

4.Models:

import tflearn

from tflearn.layers.conv import conv\_2d, max\_pool\_2d,avg\_pool\_2d, conv\_3d, max\_pool\_3d, avg\_pool\_3d

from tflearn.layers.core import input\_data, dropout, fully\_connected

from tflearn.layers.estimator import regression

from tflearn.layers.normalization import local\_response\_normalization

from tflearn.layers.merge\_ops import merge

def resnext(width, height, frame\_count, lr, output=9, model\_name = 'final\_car.model'):

net = input\_data(shape=[None, width, height, 3], name='input')

net = tflearn.conv\_2d(net, 16, 3, regularizer='L2', weight\_decay=0.0001)

net = tflearn.layers.conv.resnext\_block(net, n, 16, 32)

net = tflearn.resnext\_block(net, 1, 32, 32, downsample=True)

net = tflearn.resnext\_block(net, n-1, 32, 32)

net = tflearn.resnext\_block(net, 1, 64, 32, downsample=True)

net = tflearn.resnext\_block(net, n-1, 64, 32)

net = tflearn.batch\_normalization(net)

net = tflearn.activation(net, 'relu')

net = tflearn.global\_avg\_pool(net)

# Regression

net = tflearn.fully\_connected(net, output, activation='softmax')

opt = tflearn.Momentum(0.1, lr\_decay=0.1, decay\_step=32000, staircase=True)

net = tflearn.regression(net, optimizer=opt,

loss='categorical\_crossentropy')

model = tflearn.DNN(net,

max\_checkpoints=0, tensorboard\_verbose=0, tensorboard\_dir='log')

return model

def sentnet\_color\_2d(width, height, frame\_count, lr, output=9, model\_name = 'sentnet\_color.model'):

network = input\_data(shape=[None, width, height, 3], name='input')

network = conv\_2d(network, 96, 11, strides=4, activation='relu')

network = max\_pool\_2d(network, 3, strides=2)

network = local\_response\_normalization(network)

network = conv\_2d(network, 256, 5, activation='relu')

network = max\_pool\_2d(network, 3, strides=2)

network = local\_response\_normalization(network)

network = conv\_2d(network, 384, 3, activation='relu')

network = conv\_2d(network, 384, 3, activation='relu')

network = conv\_2d(network, 256, 3, activation='relu')

network = max\_pool\_2d(network, 3, strides=2)

network = conv\_2d(network, 256, 5, activation='relu')

network = max\_pool\_2d(network, 3, strides=2)

network = local\_response\_normalization(network)

network = conv\_2d(network, 384, 3, activation='relu')

network = conv\_2d(network, 384, 3, activation='relu')

network = conv\_2d(network, 256, 3, activation='relu')

network = max\_pool\_2d(network, 3, strides=2)

network = local\_response\_normalization(network)

network = fully\_connected(network, 4096, activation='tanh')

network = dropout(network, 0.5)

network = fully\_connected(network, 4096, activation='tanh')

network = dropout(network, 0.5)

network = fully\_connected(network, 4096, activation='tanh')

network = dropout(network, 0.5)

network = fully\_connected(network, 4096, activation='tanh')

network = dropout(network, 0.5)

network = fully\_connected(network, output, activation='softmax')

network = regression(network, optimizer='momentum',

loss='categorical\_crossentropy',

learning\_rate=lr, name='targets')

model = tflearn.DNN(network,

max\_checkpoints=0, tensorboard\_verbose=0, tensorboard\_dir='log')

return model

def inception\_v3(width, height, frame\_count, lr, output=9, model\_name = 'sentnet\_color.model'):

network = input\_data(shape=[None, width, height,3], name='input')

conv1\_7\_7 = conv\_2d(network, 64, 7, strides=2, activation='relu', name = 'conv1\_7\_7\_s2')

pool1\_3\_3 = max\_pool\_2d(conv1\_7\_7, 3,strides=2)

pool1\_3\_3 = local\_response\_normalization(pool1\_3\_3)

conv2\_3\_3\_reduce = conv\_2d(pool1\_3\_3, 64,1, activation='relu',name = 'conv2\_3\_3\_reduce')

conv2\_3\_3 = conv\_2d(conv2\_3\_3\_reduce, 192,3, activation='relu', name='conv2\_3\_3')

conv2\_3\_3 = local\_response\_normalization(conv2\_3\_3)

pool2\_3\_3 = max\_pool\_2d(conv2\_3\_3, kernel\_size=3, strides=2, name='pool2\_3\_3\_s2')

inception\_3a\_1\_1 = conv\_2d(pool2\_3\_3, 64, 1, activation='relu', name='inception\_3a\_1\_1')

inception\_3a\_3\_3\_reduce = conv\_2d(pool2\_3\_3, 96,1, activation='relu', name='inception\_3a\_3\_3\_reduce')

inception\_3a\_3\_3 = conv\_2d(inception\_3a\_3\_3\_reduce, 128,filter\_size=3, activation='relu', name = 'inception\_3a\_3\_3')

inception\_3a\_5\_5\_reduce = conv\_2d(pool2\_3\_3,16, filter\_size=1,activation='relu', name ='inception\_3a\_5\_5\_reduce' )

inception\_3a\_5\_5 = conv\_2d(inception\_3a\_5\_5\_reduce, 32, filter\_size=5, activation='relu', name= 'inception\_3a\_5\_5')

inception\_3a\_pool = max\_pool\_2d(pool2\_3\_3, kernel\_size=3, strides=1, )

inception\_3a\_pool\_1\_1 = conv\_2d(inception\_3a\_pool, 32, filter\_size=1, activation='relu', name='inception\_3a\_pool\_1\_1')

# merge the inception\_3a\_\_

inception\_3a\_output = merge([inception\_3a\_1\_1, inception\_3a\_3\_3, inception\_3a\_5\_5, inception\_3a\_pool\_1\_1], mode='concat', axis=3)

inception\_3b\_1\_1 = conv\_2d(inception\_3a\_output, 128,filter\_size=1,activation='relu', name= 'inception\_3b\_1\_1' )

inception\_3b\_3\_3\_reduce = conv\_2d(inception\_3a\_output, 128, filter\_size=1, activation='relu', name='inception\_3b\_3\_3\_reduce')

inception\_3b\_3\_3 = conv\_2d(inception\_3b\_3\_3\_reduce, 192, filter\_size=3, activation='relu',name='inception\_3b\_3\_3')

inception\_3b\_5\_5\_reduce = conv\_2d(inception\_3a\_output, 32, filter\_size=1, activation='relu', name = 'inception\_3b\_5\_5\_reduce')

inception\_3b\_5\_5 = conv\_2d(inception\_3b\_5\_5\_reduce, 96, filter\_size=5, name = 'inception\_3b\_5\_5')

inception\_3b\_pool = max\_pool\_2d(inception\_3a\_output, kernel\_size=3, strides=1, name='inception\_3b\_pool')

inception\_3b\_pool\_1\_1 = conv\_2d(inception\_3b\_pool, 64, filter\_size=1,activation='relu', name='inception\_3b\_pool\_1\_1')

#merge the inception\_3b\_\*

inception\_3b\_output = merge([inception\_3b\_1\_1, inception\_3b\_3\_3, inception\_3b\_5\_5, inception\_3b\_pool\_1\_1], mode='concat',axis=3,name='inception\_3b\_output')

pool3\_3\_3 = max\_pool\_2d(inception\_3b\_output, kernel\_size=3, strides=2, name='pool3\_3\_3')

inception\_4a\_1\_1 = conv\_2d(pool3\_3\_3, 192, filter\_size=1, activation='relu', name='inception\_4a\_1\_1')

inception\_4a\_3\_3\_reduce = conv\_2d(pool3\_3\_3, 96, filter\_size=1, activation='relu', name='inception\_4a\_3\_3\_reduce')

inception\_4a\_3\_3 = conv\_2d(inception\_4a\_3\_3\_reduce, 208, filter\_size=3, activation='relu', name='inception\_4a\_3\_3')

inception\_4a\_5\_5\_reduce = conv\_2d(pool3\_3\_3, 16, filter\_size=1, activation='relu', name='inception\_4a\_5\_5\_reduce')

inception\_4a\_5\_5 = conv\_2d(inception\_4a\_5\_5\_reduce, 48, filter\_size=5, activation='relu', name='inception\_4a\_5\_5')

inception\_4a\_pool = max\_pool\_2d(pool3\_3\_3, kernel\_size=3, strides=1, name='inception\_4a\_pool')

inception\_4a\_pool\_1\_1 = conv\_2d(inception\_4a\_pool, 64, filter\_size=1, activation='relu', name='inception\_4a\_pool\_1\_1')

inception\_4a\_output = merge([inception\_4a\_1\_1, inception\_4a\_3\_3, inception\_4a\_5\_5, inception\_4a\_pool\_1\_1], mode='concat', axis=3, name='inception\_4a\_output')

inception\_4b\_1\_1 = conv\_2d(inception\_4a\_output, 160, filter\_size=1, activation='relu', name='inception\_4a\_1\_1')

inception\_4b\_3\_3\_reduce = conv\_2d(inception\_4a\_output, 112, filter\_size=1, activation='relu', name='inception\_4b\_3\_3\_reduce')

inception\_4b\_3\_3 = conv\_2d(inception\_4b\_3\_3\_reduce, 224, filter\_size=3, activation='relu', name='inception\_4b\_3\_3')

inception\_4b\_5\_5\_reduce = conv\_2d(inception\_4a\_output, 24, filter\_size=1, activation='relu', name='inception\_4b\_5\_5\_reduce')

inception\_4b\_5\_5 = conv\_2d(inception\_4b\_5\_5\_reduce, 64, filter\_size=5, activation='relu', name='inception\_4b\_5\_5')

inception\_4b\_pool = max\_pool\_2d(inception\_4a\_output, kernel\_size=3, strides=1, name='inception\_4b\_pool')

inception\_4b\_pool\_1\_1 = conv\_2d(inception\_4b\_pool, 64, filter\_size=1, activation='relu', name='inception\_4b\_pool\_1\_1')

inception\_4b\_output = merge([inception\_4b\_1\_1, inception\_4b\_3\_3, inception\_4b\_5\_5, inception\_4b\_pool\_1\_1], mode='concat', axis=3, name='inception\_4b\_output')

inception\_4c\_1\_1 = conv\_2d(inception\_4b\_output, 128, filter\_size=1, activation='relu',name='inception\_4c\_1\_1')

inception\_4c\_3\_3\_reduce = conv\_2d(inception\_4b\_output, 128, filter\_size=1, activation='relu', name='inception\_4c\_3\_3\_reduce')

inception\_4c\_3\_3 = conv\_2d(inception\_4c\_3\_3\_reduce, 256, filter\_size=3, activation='relu', name='inception\_4c\_3\_3')

inception\_4c\_5\_5\_reduce = conv\_2d(inception\_4b\_output, 24, filter\_size=1, activation='relu', name='inception\_4c\_5\_5\_reduce')

inception\_4c\_5\_5 = conv\_2d(inception\_4c\_5\_5\_reduce, 64, filter\_size=5, activation='relu', name='inception\_4c\_5\_5')

inception\_4c\_pool = max\_pool\_2d(inception\_4b\_output, kernel\_size=3, strides=1)

inception\_4c\_pool\_1\_1 = conv\_2d(inception\_4c\_pool, 64, filter\_size=1, activation='relu', name='inception\_4c\_pool\_1\_1')

inception\_4c\_output = merge([inception\_4c\_1\_1, inception\_4c\_3\_3, inception\_4c\_5\_5, inception\_4c\_pool\_1\_1], mode='concat', axis=3,name='inception\_4c\_output')

inception\_4d\_1\_1 = conv\_2d(inception\_4c\_output, 112, filter\_size=1, activation='relu', name='inception\_4d\_1\_1')

inception\_4d\_3\_3\_reduce = conv\_2d(inception\_4c\_output, 144, filter\_size=1, activation='relu', name='inception\_4d\_3\_3\_reduce')

inception\_4d\_3\_3 = conv\_2d(inception\_4d\_3\_3\_reduce, 288, filter\_size=3, activation='relu', name='inception\_4d\_3\_3')

inception\_4d\_5\_5\_reduce = conv\_2d(inception\_4c\_output, 32, filter\_size=1, activation='relu', name='inception\_4d\_5\_5\_reduce')

inception\_4d\_5\_5 = conv\_2d(inception\_4d\_5\_5\_reduce, 64, filter\_size=5, activation='relu', name='inception\_4d\_5\_5')

inception\_4d\_pool = max\_pool\_2d(inception\_4c\_output, kernel\_size=3, strides=1, name='inception\_4d\_pool')

inception\_4d\_pool\_1\_1 = conv\_2d(inception\_4d\_pool, 64, filter\_size=1, activation='relu', name='inception\_4d\_pool\_1\_1')

inception\_4d\_output = merge([inception\_4d\_1\_1, inception\_4d\_3\_3, inception\_4d\_5\_5, inception\_4d\_pool\_1\_1], mode='concat', axis=3, name='inception\_4d\_output')

inception\_4e\_1\_1 = conv\_2d(inception\_4d\_output, 256, filter\_size=1, activation='relu', name='inception\_4e\_1\_1')

inception\_4e\_3\_3\_reduce = conv\_2d(inception\_4d\_output, 160, filter\_size=1, activation='relu', name='inception\_4e\_3\_3\_reduce')

inception\_4e\_3\_3 = conv\_2d(inception\_4e\_3\_3\_reduce, 320, filter\_size=3, activation='relu', name='inception\_4e\_3\_3')

inception\_4e\_5\_5\_reduce = conv\_2d(inception\_4d\_output, 32, filter\_size=1, activation='relu', name='inception\_4e\_5\_5\_reduce')

inception\_4e\_5\_5 = conv\_2d(inception\_4e\_5\_5\_reduce, 128, filter\_size=5, activation='relu', name='inception\_4e\_5\_5')

inception\_4e\_pool = max\_pool\_2d(inception\_4d\_output, kernel\_size=3, strides=1, name='inception\_4e\_pool')

inception\_4e\_pool\_1\_1 = conv\_2d(inception\_4e\_pool, 128, filter\_size=1, activation='relu', name='inception\_4e\_pool\_1\_1')

inception\_4e\_output = merge([inception\_4e\_1\_1, inception\_4e\_3\_3, inception\_4e\_5\_5,inception\_4e\_pool\_1\_1],axis=3, mode='concat')

pool4\_3\_3 = max\_pool\_2d(inception\_4e\_output, kernel\_size=3, strides=2, name='pool\_3\_3')

inception\_5a\_1\_1 = conv\_2d(pool4\_3\_3, 256, filter\_size=1, activation='relu', name='inception\_5a\_1\_1')

inception\_5a\_3\_3\_reduce = conv\_2d(pool4\_3\_3, 160, filter\_size=1, activation='relu', name='inception\_5a\_3\_3\_reduce')

inception\_5a\_3\_3 = conv\_2d(inception\_5a\_3\_3\_reduce, 320, filter\_size=3, activation='relu', name='inception\_5a\_3\_3')

inception\_5a\_5\_5\_reduce = conv\_2d(pool4\_3\_3, 32, filter\_size=1, activation='relu', name='inception\_5a\_5\_5\_reduce')

inception\_5a\_5\_5 = conv\_2d(inception\_5a\_5\_5\_reduce, 128, filter\_size=5, activation='relu', name='inception\_5a\_5\_5')

inception\_5a\_pool = max\_pool\_2d(pool4\_3\_3, kernel\_size=3, strides=1, name='inception\_5a\_pool')

inception\_5a\_pool\_1\_1 = conv\_2d(inception\_5a\_pool, 128, filter\_size=1,activation='relu', name='inception\_5a\_pool\_1\_1')

inception\_5a\_output = merge([inception\_5a\_1\_1, inception\_5a\_3\_3, inception\_5a\_5\_5, inception\_5a\_pool\_1\_1], axis=3,mode='concat')

inception\_5b\_1\_1 = conv\_2d(inception\_5a\_output, 384, filter\_size=1,activation='relu', name='inception\_5b\_1\_1')

inception\_5b\_3\_3\_reduce = conv\_2d(inception\_5a\_output, 192, filter\_size=1, activation='relu', name='inception\_5b\_3\_3\_reduce')

inception\_5b\_3\_3 = conv\_2d(inception\_5b\_3\_3\_reduce, 384, filter\_size=3,activation='relu', name='inception\_5b\_3\_3')

inception\_5b\_5\_5\_reduce = conv\_2d(inception\_5a\_output, 48, filter\_size=1, activation='relu', name='inception\_5b\_5\_5\_reduce')

inception\_5b\_5\_5 = conv\_2d(inception\_5b\_5\_5\_reduce,128, filter\_size=5, activation='relu', name='inception\_5b\_5\_5' )

inception\_5b\_pool = max\_pool\_2d(inception\_5a\_output, kernel\_size=3, strides=1, name='inception\_5b\_pool')

inception\_5b\_pool\_1\_1 = conv\_2d(inception\_5b\_pool, 128, filter\_size=1, activation='relu', name='inception\_5b\_pool\_1\_1')

inception\_5b\_output = merge([inception\_5b\_1\_1, inception\_5b\_3\_3, inception\_5b\_5\_5, inception\_5b\_pool\_1\_1], axis=3, mode='concat')

pool5\_7\_7 = avg\_pool\_2d(inception\_5b\_output, kernel\_size=7, strides=1)

pool5\_7\_7 = dropout(pool5\_7\_7, 0.4)

loss = fully\_connected(pool5\_7\_7, output,activation='softmax')

network = regression(loss, optimizer='momentum',

loss='categorical\_crossentropy',

learning\_rate=lr, name='targets')

model = tflearn.DNN(network,

max\_checkpoints=0, tensorboard\_verbose=0,tensorboard\_dir='log')

return model

def inception\_v3\_3d(width, height, frame\_count, lr, output=9, model\_name = 'sentnet\_color.model'):

network = input\_data(shape=[None, width, height,3, 1], name='input')

conv1\_7\_7 = conv\_3d(network, 64, 7, strides=2, activation='relu', name = 'conv1\_7\_7\_s2')

pool1\_3\_3 = max\_pool\_3d(conv1\_7\_7, 3,strides=2)

#pool1\_3\_3 = local\_response\_normalization(pool1\_3\_3)

conv2\_3\_3\_reduce = conv\_3d(pool1\_3\_3, 64,1, activation='relu',name = 'conv2\_3\_3\_reduce')

conv2\_3\_3 = conv\_3d(conv2\_3\_3\_reduce, 192,3, activation='relu', name='conv2\_3\_3')

#conv2\_3\_3 = local\_response\_normalization(conv2\_3\_3)

pool2\_3\_3 = max\_pool\_3d(conv2\_3\_3, kernel\_size=3, strides=2, name='pool2\_3\_3\_s2')

inception\_3a\_1\_1 = conv\_3d(pool2\_3\_3, 64, 1, activation='relu', name='inception\_3a\_1\_1')

inception\_3a\_3\_3\_reduce = conv\_3d(pool2\_3\_3, 96,1, activation='relu', name='inception\_3a\_3\_3\_reduce')

inception\_3a\_3\_3 = conv\_3d(inception\_3a\_3\_3\_reduce, 128,filter\_size=3, activation='relu', name = 'inception\_3a\_3\_3')

inception\_3a\_5\_5\_reduce = conv\_3d(pool2\_3\_3,16, filter\_size=1,activation='relu', name ='inception\_3a\_5\_5\_reduce' )

inception\_3a\_5\_5 = conv\_3d(inception\_3a\_5\_5\_reduce, 32, filter\_size=5, activation='relu', name= 'inception\_3a\_5\_5')

inception\_3a\_pool = max\_pool\_3d(pool2\_3\_3, kernel\_size=3, strides=1, )

inception\_3a\_pool\_1\_1 = conv\_3d(inception\_3a\_pool, 32, filter\_size=1, activation='relu', name='inception\_3a\_pool\_1\_1')

# merge the inception\_3a\_\_

inception\_3a\_output = merge([inception\_3a\_1\_1, inception\_3a\_3\_3, inception\_3a\_5\_5, inception\_3a\_pool\_1\_1], mode='concat', axis=4)

inception\_3b\_1\_1 = conv\_3d(inception\_3a\_output, 128,filter\_size=1,activation='relu', name= 'inception\_3b\_1\_1' )

inception\_3b\_3\_3\_reduce = conv\_3d(inception\_3a\_output, 128, filter\_size=1, activation='relu', name='inception\_3b\_3\_3\_reduce')

inception\_3b\_3\_3 = conv\_3d(inception\_3b\_3\_3\_reduce, 192, filter\_size=3, activation='relu',name='inception\_3b\_3\_3')

inception\_3b\_5\_5\_reduce = conv\_3d(inception\_3a\_output, 32, filter\_size=1, activation='relu', name = 'inception\_3b\_5\_5\_reduce')

inception\_3b\_5\_5 = conv\_3d(inception\_3b\_5\_5\_reduce, 96, filter\_size=5, name = 'inception\_3b\_5\_5')

inception\_3b\_pool = max\_pool\_3d(inception\_3a\_output, kernel\_size=3, strides=1, name='inception\_3b\_pool')

inception\_3b\_pool\_1\_1 = conv\_3d(inception\_3b\_pool, 64, filter\_size=1,activation='relu', name='inception\_3b\_pool\_1\_1')

#merge the inception\_3b\_\*

inception\_3b\_output = merge([inception\_3b\_1\_1, inception\_3b\_3\_3, inception\_3b\_5\_5, inception\_3b\_pool\_1\_1], mode='concat',axis=4,name='inception\_3b\_output')

pool3\_3\_3 = max\_pool\_3d(inception\_3b\_output, kernel\_size=3, strides=2, name='pool3\_3\_3')

inception\_4a\_1\_1 = conv\_3d(pool3\_3\_3, 192, filter\_size=1, activation='relu', name='inception\_4a\_1\_1')

inception\_4a\_3\_3\_reduce = conv\_3d(pool3\_3\_3, 96, filter\_size=1, activation='relu', name='inception\_4a\_3\_3\_reduce')

inception\_4a\_3\_3 = conv\_3d(inception\_4a\_3\_3\_reduce, 208, filter\_size=3, activation='relu', name='inception\_4a\_3\_3')

inception\_4a\_5\_5\_reduce = conv\_3d(pool3\_3\_3, 16, filter\_size=1, activation='relu', name='inception\_4a\_5\_5\_reduce')

inception\_4a\_5\_5 = conv\_3d(inception\_4a\_5\_5\_reduce, 48, filter\_size=5, activation='relu', name='inception\_4a\_5\_5')

inception\_4a\_pool = max\_pool\_3d(pool3\_3\_3, kernel\_size=3, strides=1, name='inception\_4a\_pool')

inception\_4a\_pool\_1\_1 = conv\_3d(inception\_4a\_pool, 64, filter\_size=1, activation='relu', name='inception\_4a\_pool\_1\_1')

inception\_4a\_output = merge([inception\_4a\_1\_1, inception\_4a\_3\_3, inception\_4a\_5\_5, inception\_4a\_pool\_1\_1], mode='concat', axis=4, name='inception\_4a\_output')

inception\_4b\_1\_1 = conv\_3d(inception\_4a\_output, 160, filter\_size=1, activation='relu', name='inception\_4a\_1\_1')

inception\_4b\_3\_3\_reduce = conv\_3d(inception\_4a\_output, 112, filter\_size=1, activation='relu', name='inception\_4b\_3\_3\_reduce')

inception\_4b\_3\_3 = conv\_3d(inception\_4b\_3\_3\_reduce, 224, filter\_size=3, activation='relu', name='inception\_4b\_3\_3')

inception\_4b\_5\_5\_reduce = conv\_3d(inception\_4a\_output, 24, filter\_size=1, activation='relu', name='inception\_4b\_5\_5\_reduce')

inception\_4b\_5\_5 = conv\_3d(inception\_4b\_5\_5\_reduce, 64, filter\_size=5, activation='relu', name='inception\_4b\_5\_5')

inception\_4b\_pool = max\_pool\_3d(inception\_4a\_output, kernel\_size=3, strides=1, name='inception\_4b\_pool')

inception\_4b\_pool\_1\_1 = conv\_3d(inception\_4b\_pool, 64, filter\_size=1, activation='relu', name='inception\_4b\_pool\_1\_1')

inception\_4b\_output = merge([inception\_4b\_1\_1, inception\_4b\_3\_3, inception\_4b\_5\_5, inception\_4b\_pool\_1\_1], mode='concat', axis=4, name='inception\_4b\_output')

inception\_4c\_1\_1 = conv\_3d(inception\_4b\_output, 128, filter\_size=1, activation='relu',name='inception\_4c\_1\_1')

inception\_4c\_3\_3\_reduce = conv\_3d(inception\_4b\_output, 128, filter\_size=1, activation='relu', name='inception\_4c\_3\_3\_reduce')

inception\_4c\_3\_3 = conv\_3d(inception\_4c\_3\_3\_reduce, 256, filter\_size=3, activation='relu', name='inception\_4c\_3\_3')

inception\_4c\_5\_5\_reduce = conv\_3d(inception\_4b\_output, 24, filter\_size=1, activation='relu', name='inception\_4c\_5\_5\_reduce')

inception\_4c\_5\_5 = conv\_3d(inception\_4c\_5\_5\_reduce, 64, filter\_size=5, activation='relu', name='inception\_4c\_5\_5')

inception\_4c\_pool = max\_pool\_3d(inception\_4b\_output, kernel\_size=3, strides=1)

inception\_4c\_pool\_1\_1 = conv\_3d(inception\_4c\_pool, 64, filter\_size=1, activation='relu', name='inception\_4c\_pool\_1\_1')

inception\_4c\_output = merge([inception\_4c\_1\_1, inception\_4c\_3\_3, inception\_4c\_5\_5, inception\_4c\_pool\_1\_1], mode='concat', axis=4,name='inception\_4c\_output')

inception\_4d\_1\_1 = conv\_3d(inception\_4c\_output, 112, filter\_size=1, activation='relu', name='inception\_4d\_1\_1')

inception\_4d\_3\_3\_reduce = conv\_3d(inception\_4c\_output, 144, filter\_size=1, activation='relu', name='inception\_4d\_3\_3\_reduce')

inception\_4d\_3\_3 = conv\_3d(inception\_4d\_3\_3\_reduce, 288, filter\_size=3, activation='relu', name='inception\_4d\_3\_3')

inception\_4d\_5\_5\_reduce = conv\_3d(inception\_4c\_output, 32, filter\_size=1, activation='relu', name='inception\_4d\_5\_5\_reduce')

inception\_4d\_5\_5 = conv\_3d(inception\_4d\_5\_5\_reduce, 64, filter\_size=5, activation='relu', name='inception\_4d\_5\_5')

inception\_4d\_pool = max\_pool\_3d(inception\_4c\_output, kernel\_size=3, strides=1, name='inception\_4d\_pool')

inception\_4d\_pool\_1\_1 = conv\_3d(inception\_4d\_pool, 64, filter\_size=1, activation='relu', name='inception\_4d\_pool\_1\_1')

inception\_4d\_output = merge([inception\_4d\_1\_1, inception\_4d\_3\_3, inception\_4d\_5\_5, inception\_4d\_pool\_1\_1], mode='concat', axis=4, name='inception\_4d\_output')

inception\_4e\_1\_1 = conv\_3d(inception\_4d\_output, 256, filter\_size=1, activation='relu', name='inception\_4e\_1\_1')

inception\_4e\_3\_3\_reduce = conv\_3d(inception\_4d\_output, 160, filter\_size=1, activation='relu', name='inception\_4e\_3\_3\_reduce')

inception\_4e\_3\_3 = conv\_3d(inception\_4e\_3\_3\_reduce, 320, filter\_size=3, activation='relu', name='inception\_4e\_3\_3')

inception\_4e\_5\_5\_reduce = conv\_3d(inception\_4d\_output, 32, filter\_size=1, activation='relu', name='inception\_4e\_5\_5\_reduce')

inception\_4e\_5\_5 = conv\_3d(inception\_4e\_5\_5\_reduce, 128, filter\_size=5, activation='relu', name='inception\_4e\_5\_5')

inception\_4e\_pool = max\_pool\_3d(inception\_4d\_output, kernel\_size=3, strides=1, name='inception\_4e\_pool')

inception\_4e\_pool\_1\_1 = conv\_3d(inception\_4e\_pool, 128, filter\_size=1, activation='relu', name='inception\_4e\_pool\_1\_1')

inception\_4e\_output = merge([inception\_4e\_1\_1, inception\_4e\_3\_3, inception\_4e\_5\_5,inception\_4e\_pool\_1\_1],axis=4, mode='concat')

pool4\_3\_3 = max\_pool\_3d(inception\_4e\_output, kernel\_size=3, strides=2, name='pool\_3\_3')

inception\_5a\_1\_1 = conv\_3d(pool4\_3\_3, 256, filter\_size=1, activation='relu', name='inception\_5a\_1\_1')

inception\_5a\_3\_3\_reduce = conv\_3d(pool4\_3\_3, 160, filter\_size=1, activation='relu', name='inception\_5a\_3\_3\_reduce')

inception\_5a\_3\_3 = conv\_3d(inception\_5a\_3\_3\_reduce, 320, filter\_size=3, activation='relu', name='inception\_5a\_3\_3')

inception\_5a\_5\_5\_reduce = conv\_3d(pool4\_3\_3, 32, filter\_size=1, activation='relu', name='inception\_5a\_5\_5\_reduce')

inception\_5a\_5\_5 = conv\_3d(inception\_5a\_5\_5\_reduce, 128, filter\_size=5, activation='relu', name='inception\_5a\_5\_5')

inception\_5a\_pool = max\_pool\_3d(pool4\_3\_3, kernel\_size=3, strides=1, name='inception\_5a\_pool')

inception\_5a\_pool\_1\_1 = conv\_3d(inception\_5a\_pool, 128, filter\_size=1,activation='relu', name='inception\_5a\_pool\_1\_1')

inception\_5a\_output = merge([inception\_5a\_1\_1, inception\_5a\_3\_3, inception\_5a\_5\_5, inception\_5a\_pool\_1\_1], axis=4,mode='concat')

inception\_5b\_1\_1 = conv\_3d(inception\_5a\_output, 384, filter\_size=1,activation='relu', name='inception\_5b\_1\_1')

inception\_5b\_3\_3\_reduce = conv\_3d(inception\_5a\_output, 192, filter\_size=1, activation='relu', name='inception\_5b\_3\_3\_reduce')

inception\_5b\_3\_3 = conv\_3d(inception\_5b\_3\_3\_reduce, 384, filter\_size=3,activation='relu', name='inception\_5b\_3\_3')

inception\_5b\_5\_5\_reduce = conv\_3d(inception\_5a\_output, 48, filter\_size=1, activation='relu', name='inception\_5b\_5\_5\_reduce')

inception\_5b\_5\_5 = conv\_3d(inception\_5b\_5\_5\_reduce,128, filter\_size=5, activation='relu', name='inception\_5b\_5\_5' )

inception\_5b\_pool = max\_pool\_3d(inception\_5a\_output, kernel\_size=3, strides=1, name='inception\_5b\_pool')

inception\_5b\_pool\_1\_1 = conv\_3d(inception\_5b\_pool, 128, filter\_size=1, activation='relu', name='inception\_5b\_pool\_1\_1')

inception\_5b\_output = merge([inception\_5b\_1\_1, inception\_5b\_3\_3, inception\_5b\_5\_5, inception\_5b\_pool\_1\_1], axis=4, mode='concat')

pool5\_7\_7 = avg\_pool\_3d(inception\_5b\_output, kernel\_size=7, strides=1)

pool5\_7\_7 = dropout(pool5\_7\_7, 0.4)

loss = fully\_connected(pool5\_7\_7, output,activation='softmax')

network = regression(loss, optimizer='momentum',

loss='categorical\_crossentropy',

learning\_rate=lr, name='targets')

model = tflearn.DNN(network, checkpoint\_path=model\_name,

max\_checkpoints=1, tensorboard\_verbose=0,tensorboard\_dir='log')

return model

def sentnet\_LSTM\_gray(width, height, frame\_count, lr, output=9):

network = input\_data(shape=[None, width, height], name='input')

#network = tflearn.input\_data(shape=[None, 28, 28], name='input')

network = tflearn.lstm(network, 128, return\_seq=True)

network = tflearn.lstm(network, 128)

network = tflearn.fully\_connected(network, 9, activation='softmax')

network = tflearn.regression(network, optimizer='adam',

loss='categorical\_crossentropy', name="output1")

model = tflearn.DNN(network, checkpoint\_path='model\_lstm',

max\_checkpoints=1, tensorboard\_verbose=0, tensorboard\_dir='log')

return model

def sentnet\_color(width, height, frame\_count, lr, output=9, model\_name = 'sentnet\_color.model'):

network = input\_data(shape=[None, width, height,3, 1], name='input')

network = conv\_3d(network, 96, 11, strides=4, activation='relu')

network = max\_pool\_3d(network, 3, strides=2)

#network = local\_response\_normalization(network)

network = conv\_3d(network, 256, 5, activation='relu')

network = max\_pool\_3d(network, 3, strides=2)

#network = local\_response\_normalization(network)

network = conv\_3d(network, 384, 3, activation='relu')

network = conv\_3d(network, 384, 3, activation='relu')

network = conv\_3d(network, 256, 3, activation='relu')

network = max\_pool\_3d(network, 3, strides=2)

network = conv\_3d(network, 256, 5, activation='relu')

network = max\_pool\_3d(network, 3, strides=2)

#network = local\_response\_normalization(network)

network = conv\_3d(network, 384, 3, activation='relu')

network = conv\_3d(network, 384, 3, activation='relu')

network = conv\_3d(network, 256, 3, activation='relu')

network = max\_pool\_3d(network, 3, strides=2)

#network = local\_response\_normalization(network)

network = fully\_connected(network, 4096, activation='tanh')

network = dropout(network, 0.5)

network = fully\_connected(network, 4096, activation='tanh')

network = dropout(network, 0.5)

network = fully\_connected(network, 4096, activation='tanh')

network = dropout(network, 0.5)

network = fully\_connected(network, 4096, activation='tanh')

network = dropout(network, 0.5)

network = fully\_connected(network, output, activation='softmax')

network = regression(network, optimizer='momentum',

loss='categorical\_crossentropy',

learning\_rate=lr, name='targets')

model = tflearn.DNN(network, checkpoint\_path=model\_name,

max\_checkpoints=1, tensorboard\_verbose=0, tensorboard\_dir='log')

return model

def sentnet\_frames(width, height, frame\_count, lr, output=9):

network = input\_data(shape=[None, width, height,frame\_count, 1], name='input')

network = conv\_3d(network, 96, 11, strides=4, activation='relu')

network = max\_pool\_3d(network, 3, strides=2)

#network = local\_response\_normalization(network)

network = conv\_3d(network, 256, 5, activation='relu')

network = max\_pool\_3d(network, 3, strides=2)

#network = local\_response\_normalization(network)

network = conv\_3d(network, 384, 3, activation='relu')

network = conv\_3d(network, 384, 3, activation='relu')

network = conv\_3d(network, 256, 3, activation='relu')

network = max\_pool\_3d(network, 3, strides=2)

network = conv\_3d(network, 256, 5, activation='relu')

network = max\_pool\_3d(network, 3, strides=2)

#network = local\_response\_normalization(network)

network = conv\_3d(network, 384, 3, activation='relu')

network = conv\_3d(network, 384, 3, activation='relu')

network = conv\_3d(network, 256, 3, activation='relu')

network = max\_pool\_3d(network, 3, strides=2)

#network = local\_response\_normalization(network)

network = fully\_connected(network, 4096, activation='tanh')

network = dropout(network, 0.5)

network = fully\_connected(network, 4096, activation='tanh')

network = dropout(network, 0.5)

network = fully\_connected(network, 4096, activation='tanh')

network = dropout(network, 0.5)

network = fully\_connected(network, 4096, activation='tanh')

network = dropout(network, 0.5)

network = fully\_connected(network, output, activation='softmax')

network = regression(network, optimizer='momentum',

loss='categorical\_crossentropy',

learning\_rate=lr, name='targets')

model = tflearn.DNN(network, checkpoint\_path='model\_alexnet',

max\_checkpoints=1, tensorboard\_verbose=0, tensorboard\_dir='log')

return model

def sentnet2(width, height, frame\_count, lr, output=9):

network = input\_data(shape=[None, width, height, frame\_count, 1], name='input')

network = conv\_3d(network, 96, 11, strides=4, activation='relu')

network = max\_pool\_3d(network, 3, strides=2)

#network = local\_response\_normalization(network)

network = conv\_3d(network, 256, 5, activation='relu')

network = max\_pool\_3d(network, 3, strides=2)

#network = local\_response\_normalization(network)

network = conv\_3d(network, 384, 3, activation='relu')

network = conv\_3d(network, 384, 3, activation='relu')

network = conv\_3d(network, 256, 3, activation='relu')

network = max\_pool\_3d(network, 3, strides=2)

#network = local\_response\_normalization(network)

network = fully\_connected(network, 4096, activation='tanh')

network = dropout(network, 0.5)

network = fully\_connected(network, 4096, activation='tanh')

network = dropout(network, 0.5)

network = fully\_connected(network, 3, activation='softmax')

network = regression(network, optimizer='momentum',

loss='categorical\_crossentropy',

learning\_rate=lr, name='targets')

model = tflearn.DNN(network, checkpoint\_path='model\_alexnet',

max\_checkpoints=1, tensorboard\_verbose=0, tensorboard\_dir='log')

return model

def sentnet(width, height, frame\_count, lr, output=9):

network = input\_data(shape=[None, width, height, frame\_count, 1], name='input')

network = conv\_3d(network, 96, 11, strides=4, activation='relu')

network = avg\_pool\_3d(network, 3, strides=2)

#network = local\_response\_normalization(network)

network = conv\_3d(network, 256, 5, activation='relu')

network = avg\_pool\_3d(network, 3, strides=2)

#network = local\_response\_normalization(network)

network = conv\_3d(network, 384, 3, activation='relu')

network = conv\_3d(network, 384, 3, activation='relu')

network = conv\_3d(network, 256, 3, activation='relu')

network = max\_pool\_3d(network, 3, strides=2)

network = conv\_3d(network, 256, 5, activation='relu')

network = avg\_pool\_3d(network, 3, strides=2)

#network = local\_response\_normalization(network)

network = conv\_3d(network, 384, 3, activation='relu')

network = conv\_3d(network, 384, 3, activation='relu')

network = conv\_3d(network, 256, 3, activation='relu')

network = avg\_pool\_3d(network, 3, strides=2)

#network = local\_response\_normalization(network)

network = fully\_connected(network, 4096, activation='tanh')

network = dropout(network, 0.5)

network = fully\_connected(network, 4096, activation='tanh')

network = dropout(network, 0.5)

network = fully\_connected(network, 4096, activation='tanh')

network = dropout(network, 0.5)

network = fully\_connected(network, 4096, activation='tanh')

network = dropout(network, 0.5)

network = fully\_connected(network, output, activation='softmax')

network = regression(network, optimizer='momentum',

loss='categorical\_crossentropy',

learning\_rate=lr, name='targets')

model = tflearn.DNN(network, checkpoint\_path='model\_alexnet',

max\_checkpoints=1, tensorboard\_verbose=0, tensorboard\_dir='log')

return model

def alexnet2(width, height, lr, output=3):

network = input\_data(shape=[None, width, height, 1], name='input')

network = conv\_2d(network, 96, 11, strides=4, activation='relu')

network = max\_pool\_2d(network, 3, strides=2)

network = local\_response\_normalization(network)

network = conv\_2d(network, 256, 5, activation='relu')

network = max\_pool\_2d(network, 3, strides=2)

network = local\_response\_normalization(network)

network = conv\_2d(network, 384, 3, activation='relu')

network = conv\_2d(network, 384, 3, activation='relu')

network = conv\_2d(network, 256, 3, activation='relu')

network = max\_pool\_2d(network, 3, strides=2)

network = conv\_2d(network, 256, 5, activation='relu')

network = max\_pool\_2d(network, 3, strides=2)

network = local\_response\_normalization(network)

network = conv\_2d(network, 384, 3, activation='relu')

network = conv\_2d(network, 384, 3, activation='relu')

network = conv\_2d(network, 256, 3, activation='relu')

network = max\_pool\_2d(network, 3, strides=2)

network = local\_response\_normalization(network)

network = fully\_connected(network, 4096, activation='tanh')

network = dropout(network, 0.5)

network = fully\_connected(network, 4096, activation='tanh')

network = dropout(network, 0.5)

network = fully\_connected(network, 4096, activation='tanh')

network = dropout(network, 0.5)

network = fully\_connected(network, 4096, activation='tanh')

network = dropout(network, 0.5)

network = fully\_connected(network, output, activation='softmax')

network = regression(network, optimizer='momentum',

loss='categorical\_crossentropy',

learning\_rate=lr, name='targets')

model = tflearn.DNN(network, checkpoint\_path='model\_alexnet',

max\_checkpoints=1, tensorboard\_verbose=0, tensorboard\_dir='log')

return model

def sentnet\_v0(width, height, frame\_count, lr, output=9):

network = input\_data(shape=[None, width, height, frame\_count, 1], name='input')

network = conv\_3d(network, 96, 11, strides=4, activation='relu')

network = max\_pool\_3d(network, 3, strides=2)

#network = local\_response\_normalization(network)

network = conv\_3d(network, 256, 5, activation='relu')

network = max\_pool\_3d(network, 3, strides=2)

#network = local\_response\_normalization(network)

network = conv\_3d(network, 384, 3, 3, activation='relu')

network = conv\_3d(network, 384, 3, 3, activation='relu')

network = conv\_3d(network, 256, 3, 3, activation='relu')

network = max\_pool\_3d(network, 3, strides=2)

#network = local\_response\_normalization(network)

network = fully\_connected(network, 4096, activation='tanh')

network = dropout(network, 0.5)

network = fully\_connected(network, 4096, activation='tanh')

network = dropout(network, 0.5)

network = fully\_connected(network, output, activation='softmax')

network = regression(network, optimizer='momentum',

loss='categorical\_crossentropy',

learning\_rate=lr, name='targets')

model = tflearn.DNN(network, checkpoint\_path='model\_alexnet',

max\_checkpoints=1, tensorboard\_verbose=0, tensorboard\_dir='log')

return model

def alexnet(width, height, lr, output=3):

network = input\_data(shape=[None, width, height, 1], name='input')

network = conv\_2d(network, 96, 11, strides=4, activation='relu')

network = max\_pool\_2d(network, 3, strides=2)

network = local\_response\_normalization(network)

network = conv\_2d(network, 256, 5, activation='relu')

network = max\_pool\_2d(network, 3, strides=2)

network = local\_response\_normalization(network)

network = conv\_2d(network, 384, 3, activation='relu')

network = conv\_2d(network, 384, 3, activation='relu')

network = conv\_2d(network, 256, 3, activation='relu')

network = max\_pool\_2d(network, 3, strides=2)

network = local\_response\_normalization(network)

network = fully\_connected(network, 4096, activation='tanh')

network = dropout(network, 0.5)

network = fully\_connected(network, 4096, activation='tanh')

network = dropout(network, 0.5)

network = fully\_connected(network, output, activation='softmax')

network = regression(network, optimizer='momentum',

loss='categorical\_crossentropy',

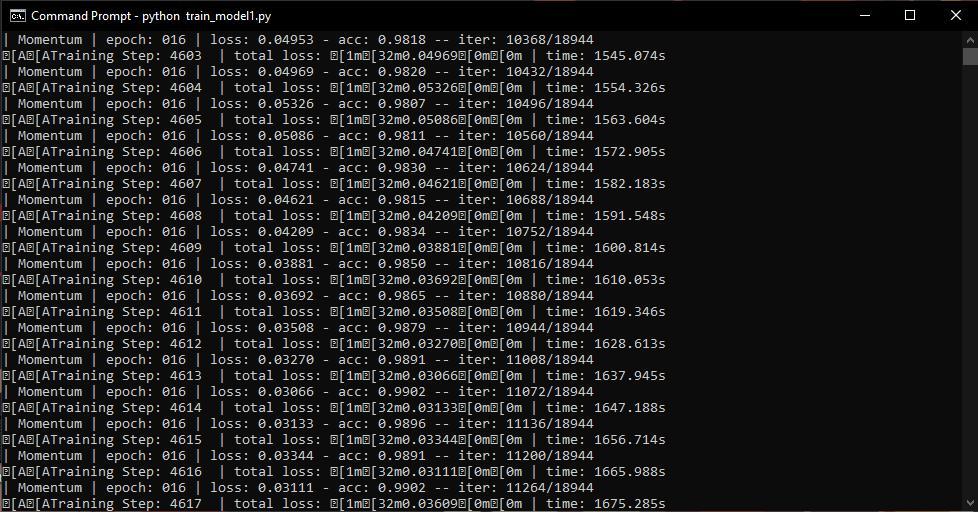
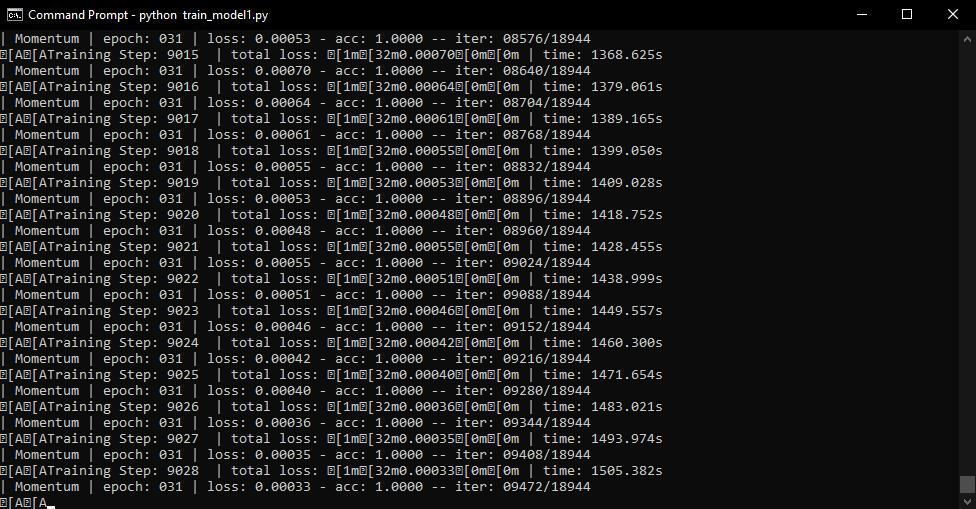
learning\_rate=lr, name='targets')

model = tflearn.DNN(network, checkpoint\_path='model\_alexnet',

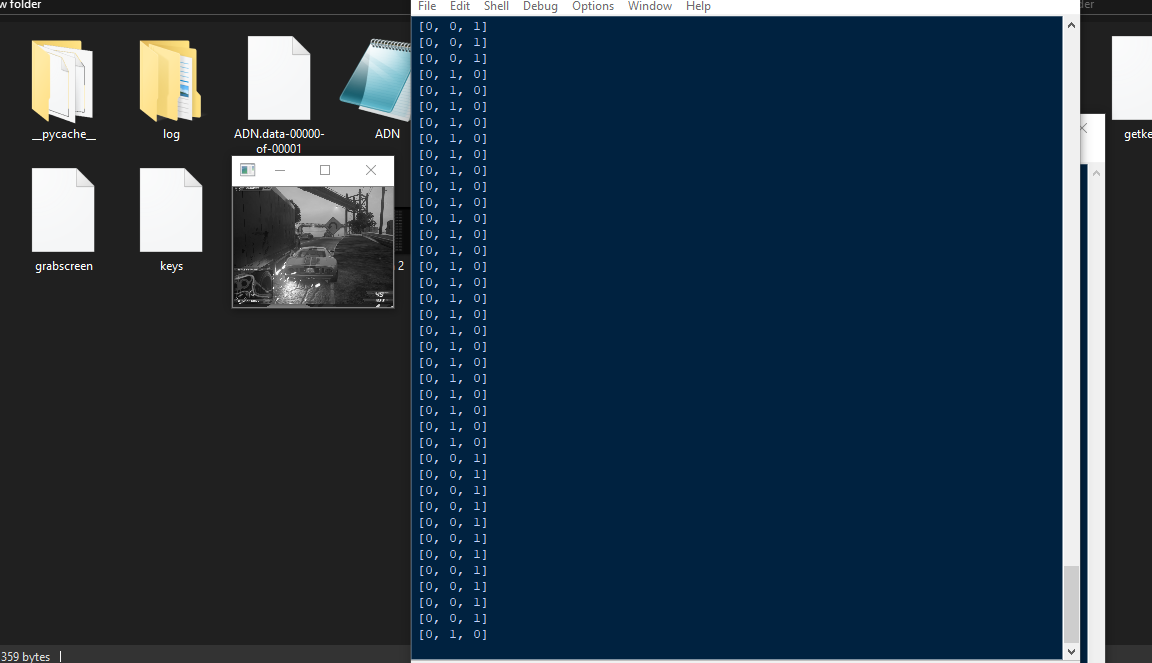
max\_checkpoints=1, tensorboard\_verbose=0, tensorboard\_dir='log')

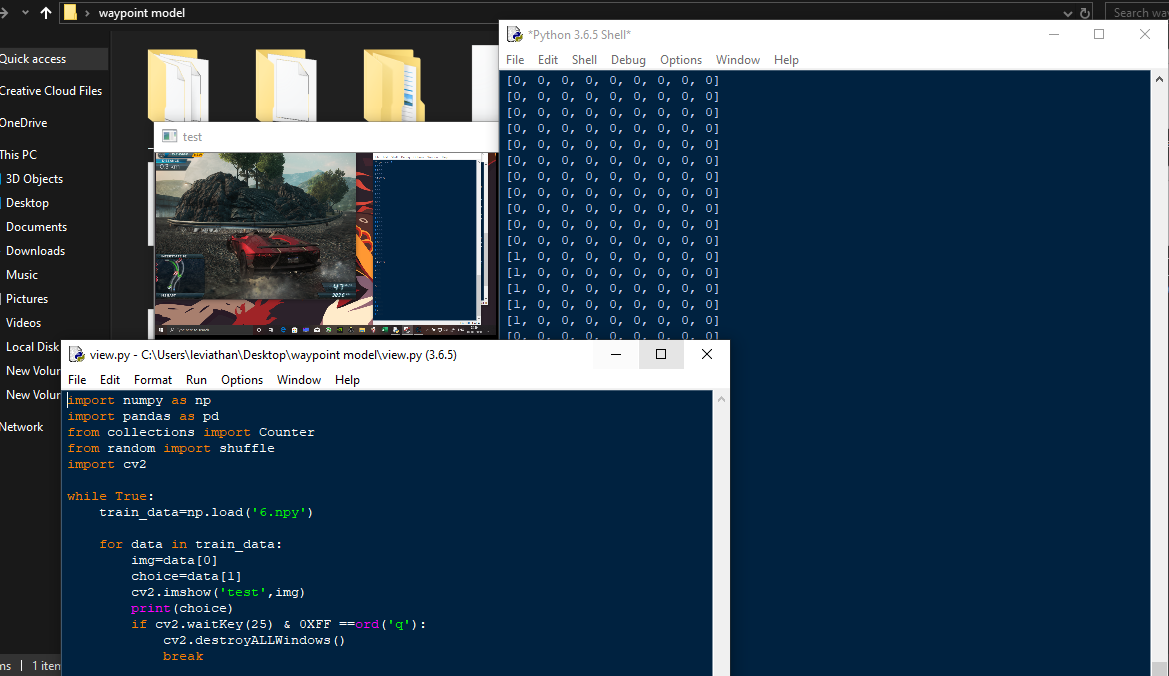
return model

Images:









Plagiarism:

