Manarat International University

Department of **Computer Science & Engineering**

Neural Networks and Fuzzy Systems (CSE-433)

& Computer Vision & Robotics (CSE-437)

Project Report

Problem Tile

CIFAR-10 - Object Recognition in Images

Team Info

- **Name of Our Team: Friends**
- **Kaggle & Github Repository links of project:**
- **Contestants Name & Student ID:**

➢ **Minhazul Zannat**	**:: 1640CSE00466**
➢ **Ashrafujjaman**	**:: 1640CSE00537**

Problem Statement - (Introduction)

The CIFAR-10 dataset is a collection of images that are commonly used to train computer vision algorithms. It is one of the most widely used datasets for deep learning research. As **Kaggle** says, the CIFAR-10 dataset contains 60,000 32x32 RGB color images in 10 different classes which are airplanes, cars, birds, cats, deer, dogs, frogs, horses, ships, and trucks. There are 6,000 images of each class.
The competition is to predict the labels of total 300,000 images on this dataset using only 60,000 labeled images for training.

Actually there are 50,000 labeled images for training with 5,000 of each class.
What's the goal of it?
Its goal is to introduce us with the computer vision algorithms and also with deep learning research as well as teach us the way to process data and handle them on model training so that the model can act what is desired to do. It helps us to learn

the way of deep learning and also image handling for that. It's a beginner level project so we only face an introduction of the concepts and so we **shouldn't go over the moon** thinking that we get a big theater after finishing it. It's one of my teacher's remark and I love it most.

Now, on behalf of my team I, Minhazul Zannat introduce you with the steps and works of us which we have taken already to make our project finished. In bellow I am going to explain how we perform the steps in a quick briefing. For more detail you can visit our Github link which is mentioned already.

Data Preprocessing

As the data-set is well established because of being a beginner level competition, so we don't have to do a large number of preprocessing on images. But yes, we have performed a little bit of that on data and they are:

❖ Normalization:

Normalization is a database design technique which organizes tables in a manner that reduces redundancy and dependency of data. It divides larger tables to smaller tables and links them using relationships. We have done that on these manner.

1. **Get Pixel's range in (-1** to **1) from (0** to **255):**

   Method: ((pixel / 255) – 0.5) * 2.0

2. **Pixel Operation using "mean" & "standard deviation":**

   Method: (data - mean)/ (std + 1e-7)

   Here,

   data = train or test images

   mean = mean of train images

   std = standard deviation of train images

Our project's code of it is in: manage/src/preprocess.py

❖ Augmentation:

Our contest is to achieve better understanding of deep learning. But it's a must to have a lot of data for it. Though we have a lot of it but it's not enough. For example for a medium size neural network, 50,000 train images are too much little. For so we have used data augmentation.

Image data augmentation is a technique that can be used to artificially expand the size of a training dataset by creating modified versions of images in the dataset. Image data augmentation is used to expand the training dataset in order to improve the performance and ability of the model to generalize.

Our used techniques, which were implemented on images are:

- **shear_range=0.2,**
- **zoom_range=0.2,**
- **rotation_range=15,**
- **width_shift_range=0.1,**
- **height_shift_range=0.1,**
- **fill_mode='nearest',**
- **horizontal_flip=True,**

Details of these codes can be found in: ImageDataGenerator class of keras.
Our project's code of it is in: manage/src/train.py

Network Architecture

We have used many Architecture for our project. We mainly have created multiple models and then combine their results. Some models which keeps our heart with full of energy in this long work time (around 30days work) are introduced bellow.

❖ Baseline model:

As the first try of walking, always inspires us to stand again and again even after a lot of failures, the baseline model, which was given by our course teacher makes the first push for us towards the success in the project. But we have done some modification there and our modified baseline model becomes as shown in the file: "manage/src/my_model_baseline.py". We have got 0.8144 validation accuracy from it on training with:

a. **180 epochs**
b. **200 batch size**
c. **LeakyReLU(alpha=0.01) activation function**
d. **[32, 32, 64, 64] filter size of conv2D (convolution layer)**
e. **[0.20, 0.20, 0.30] dropout**

As sequence (top to down) in file and got 0.81270 score in Kaggle.

❖ Simple cnn 90%+:

"Simple cnn 90%+" – v1

It's really an honor to get the model in Kaggle discussion. It has changed our base inspiration at a high level. We also have done some modification in it and the file is in our project as: "manage/src/my_model_90_des_861.py". We have got 0.8567 validation accuracy from it on training with:

a. **57 epochs**
b. **200 batch size**
c. **LeakyReLU(alpha=0.01) activation function**
d. **[32, 32, 64, 64, 128, 128] filter size of conv2D (convolution layer)**
e. **[0.20, 0.25, 0.30, 0.35] dropout**

As sequence (top to down) in file and got 0.86160 score in Kaggle.

"Simple cnn 90%+" -- v2

In later we have modified it again and then it gives us our best result (using single model) and the file is: "manage/src/my_model_90_des-x_873.py". We have got the 0. 8711 validation accuracy from it on training with:

a. **202 epochs**
b. **200 batch size**
c. **LeakyReLU(alpha=0.01) activation function**
d. **In it we have deleted the mid-level dense operation, which was used in previous version (upper model) of it.**
e. **[32, 32, 64, 64, 128, 128] filter size of conv2D (convolution layer)**
f. **[0.15, 0.20, 0.25] dropout**

As sequence (top to down) in file and got 0. 87390 score in Kaggle.

❖ Googlenet – Resnet concepts:

We have fond another model in keras discussion. It has got our interest a lot, as it looks like a different approach from the approaches we have discovered till then. So we have used to visit the Github link which is given in there. We have named it **X-approach**. At that time we were introduced with Googlenet-Resnet concepts in our class and I was so sick to learn more about them. After coming back home I have searched about them in most where and an idea just cross over my mind. We have implemented the Googlenet-Resnet concepts with the recently found model based on **X-approach**. We then have created our own model followed by the Googlenet – resnet concepts and implemented by using the concept of the model which contains the **X-approach**. That was the first model of us what we have created and the file is in our project: "manage/src/my_model_marge_net.py". As the competition time was too short then (3days left) because of spending a lot of time to create it without any base. So we have to run the model for a few time. We have got the 0. 8562 validation accuracy from it on training with:

a. **47 epochs**
b. **140 batch size**
c. **LeakyReLU(alpha=0.01) activation function for main function block**
d. **Relu activation function for resnet function block**
e. **We have used Googlenet to shape the Conv2D layers of resnet block.**
f. **It needs too much time to do one epoch (30 minutes for 1st one). So we can't spend more time to better it.**
g. **[16, 32, 64] filter size of conv2D (convolution layer)**
h. **[0.25, 0.30, 0.35] dropout**

As sequence (top to down) in file and got 0. 86970 score in Kaggle.

This was the model for which we get the top position. You may think "But how? Its result is not that one for what you have the top position." Let's keep this thing a side for now. We will continue surely about this after sometime.

❖ Approach of model-X:

We call the model-X which approach we have introduced already in Googlenet – Resnet block (the **X-approach**). It's the model at which we have got an interest for its different approach and which concept we have used in our Googlenet-resnet merge model. The model we have found in keras discussion. We have changed it a little bit (like: change the function's components) and it was now as like in our project has "manage/src/my_model_x_full_lrelu.py". We have run it just one day before our deadline of competition and it also requires a lot of time on training (29 minutes for 1st epoch). But it couldn't give us a better result in time. As we have a lot of interest in it, we have continued our epoch processing on it even when the competition was over. And for a disrespectful work of us, the result of it is still unknown as we have forgotten to store the result of it (as the competition was finished then, so we didn't show much importance in storing it). But it gives us a new teach that sometime we should care about the tortoise even when we are at the finish line. We will again talk about our mistakes and its result after a while. The parameters of it were:

a. **49 epochs**
b. **140 batch size**
c. **LeakyReLU(alpha=0.01) activation function**
d. **[16, 32, 64] filter size of conv2D (convolution layer)**
e. **[0.20, 0.25, 0.30] dropout.**

As sequence (top to down) in file and we have lost the submission file of it which is necessary to get the Kaggle score of that model (we have programmed as like running process can replace the previous process's submission file. So, when we have run a process of saving submission file after its completion then that has removed the result file of it).

These are some important parameters used for the individual models. But there are also some others too which can easily be found while getting into the project.

Training Procedure:

We have already done briefing about most of our training procedures of our individual models (like how much epoch, which batch size, which activation function, filter size, dropout size) and have to do so as out different models have different parameters. So we would like to keep the individual perimeters with their own. But still there are some common parameters as well as and they are:

a. **validation_split=0.25** -> can find in: "manage/src/train.py"
b. **img_shape = 32 * 32 * 3** -> can find in: "manage/config.py"
   & in some models too
c. **model_optimizer =** (-> can find in some models)
   Adam (lr=0.01, beta_1=0.9, beta_2=0.999, epsilon=1e-08, decay=0.0)

These are also some other common parameters which can be clarified while visiting the project. We want to introduce some of our important steps for which we have got the position and about those for which I have said "We will continue" in above.

❖ Merge models

This was the trick for which we have got the golden batch 🥇 from our loving sir. We have merged our models using **max voting**. The models which have given us the top validation accuracy till then, have been merged together as like:

"0.8567", "0.8711", "0.8562"; where default was "0.8711"

Means: We have merged

"Simple cnn 90%+" -- v1,
"Simple cnn 90%+" -- v2,
"Googlenet-resnet concept"

Where "Simple cnn 90%+ -- v2" – was the default as it has the top most score till then. And it has given us the result "0. 88910". The file using which we have merged the models is in: "manage/src/marge_models.py".

❖ Merge Submissions

Though our result has introduced already, we want to brief a little bit more. As I have said before in "**Approach of model-X**", the model was running even after the competition has finished. It has finished it's epoch after 6 hours of our competition over. Then for having a special interest on it I have merged the submission of it with the submission files of the previous models. The file using which we have merged the submissions was created before competition has finished and we have created that on this purpose only and only for this model too. We have merged our submissions using **max voting** again.

The file is in: "manage/src/ marge_submissions.py". The submissions which have given us the top scores in Kaggle till then have been merged together as like:

"submission_mz.csv", "0.8711", "0.88910"; where default was "0.88910"

Means: We have merged the submission files of

"Merge Submissions",
"Simple cnn 90%+" -- v2,
"Merge models"

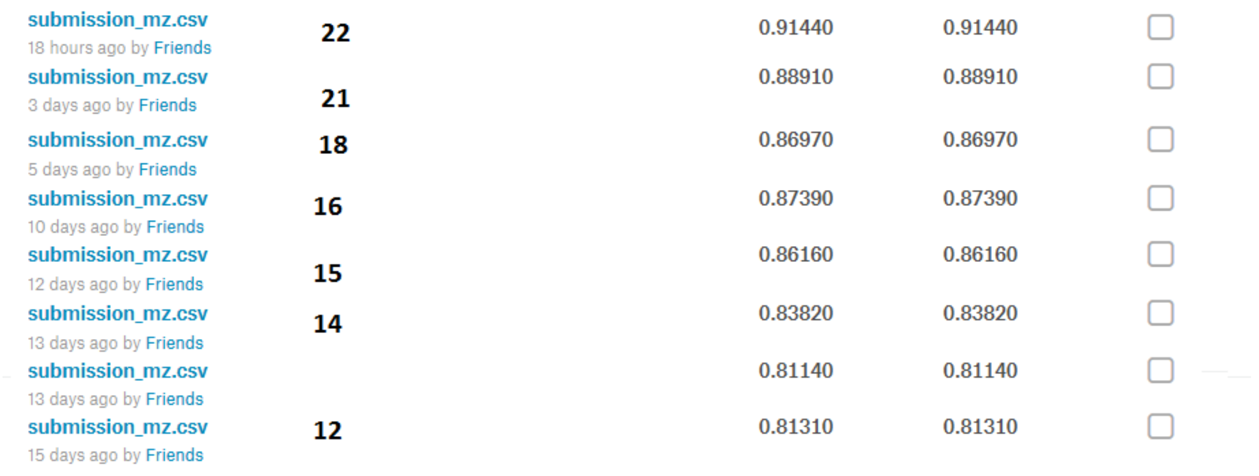
Where "Merge models" – was the default as it has the top most score till then. And it gives us the result "0. 91440". As it's the result after our competition has over so that we have kept our score as "0. 88910" (for competition).

I only want to share this for clarifying the "manage/src/ marge_submissions.py" file's use and reason of showing our result as "0. 91440" in Github though our competition result is "0. 88910".

For more details you can check a file called “steps_description.txt”. We have kept some important notes for our models in order to keep tracking on model’s performance. The file is in: “manage/dataset/steps_description.txt”.

Result:

We have already known about our final score in Kaggle. But I want to present that in an official manner. And also some of our other submissions are given too.
As the report becomes too broad already I have reduced the screenshot shape of the results. So we are presenting our **score for competition** here:
Final Submission score of us: “0. 88910”
Total Submissions: 21 times



submission_mz.csv 18 hours ago by Friends	**22**	0.91440	0.91440	☐
submission_mz.csv 3 days ago by Friends	**21**	0.88910	0.88910	☐
submission_mz.csv 5 days ago by Friends	**18**	0.86970	0.86970	☐
submission_mz.csv 10 days ago by Friends	**16**	0.87390	0.87390	☐
submission_mz.csv 12 days ago by Friends	**15**	0.86160	0.86160	☐
submission_mz.csv 13 days ago by Friends	**14**	0.83820	0.83820	☐
submission_mz.csv 13 days ago by Friends		0.81140	0.81140	☐
submission_mz.csv 15 days ago by Friends	**12**	0.81310	0.81310	☐

Information about Operating System & Environment:

Our Project was run in one laptop only. So, we describe as:

PC model: **Acer Aspire V3-572P**
OS **(Dual boot):** **Windows_8.1** & **Kali-Linux_13.7**
(Most of the typing was in Windows & code was run in Linux)
Ram size: **6GB**

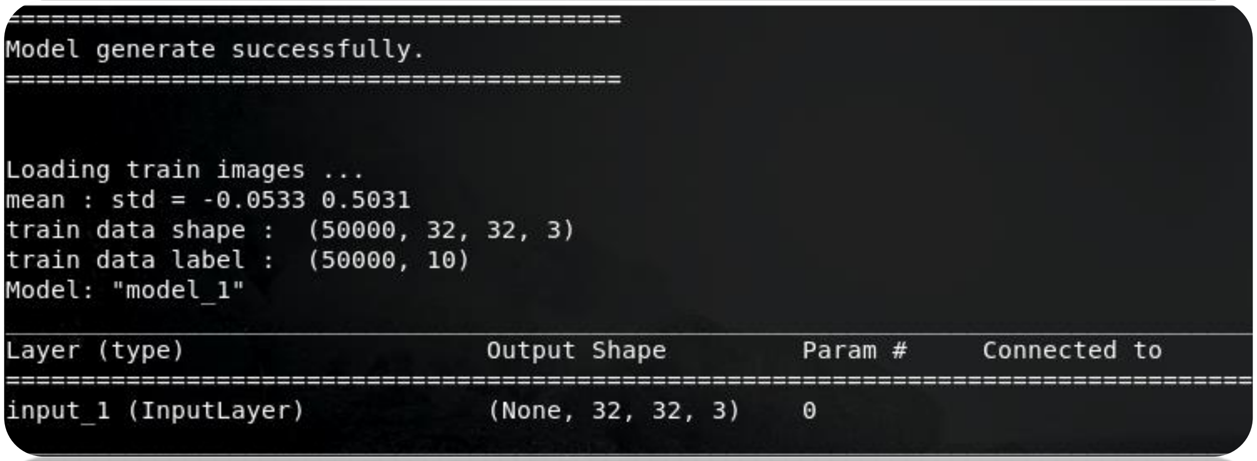
==

System Type: **64-bit both OS, x64-bit based Processor**
Linux Terminal: **Genome v-3.26.2**
Programming Language: **Python - 3.6.3** (Using virtualenv in Linux)
Text Editor: **Sublime-text-3** (In Win8.1) & **VS-code** (In Linux)

<u>Some screenshots of our models at training:</u>

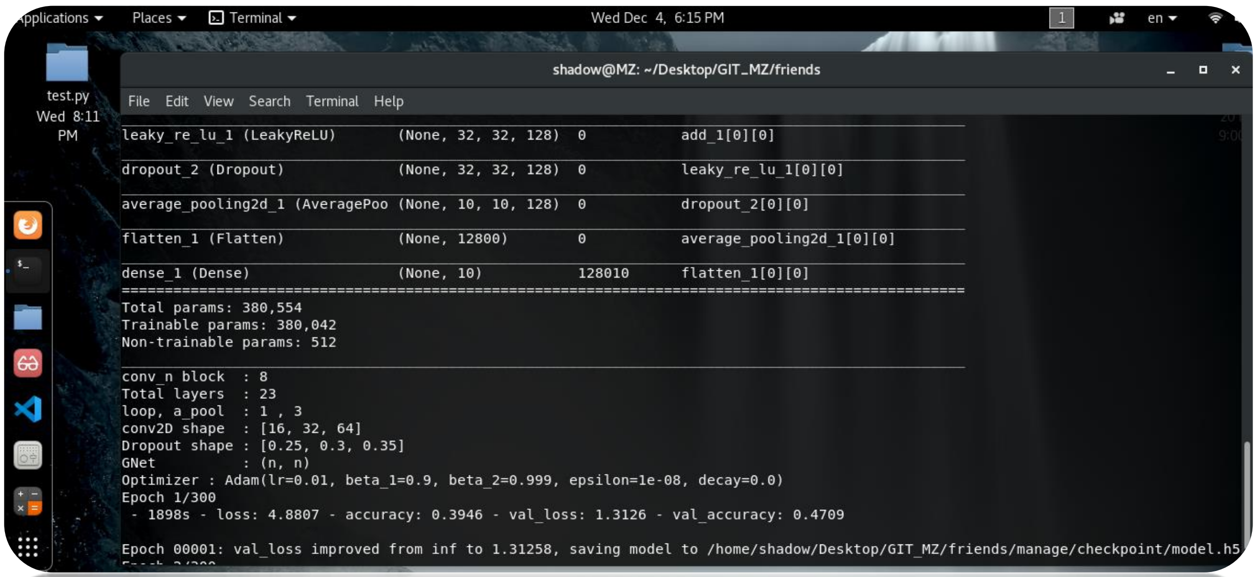
We only show those models which are caries some detail of them. The model architecture can easily be found by only running the models. And they are too much broad to capture in one screenshot so we are skipping them for showing in here. We also like to present some other parameters too which are important as much that we think:

Mean & Standard Deviation of images



```
=========================================
Model generate successfully.
=========================================


Loading train images ...
mean : std = -0.0533 0.5031
train data shape :  (50000, 32, 32, 3)
train data label :  (50000, 10)
Model: "model_1"
________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
================================================================================
input_1 (InputLayer)            (None, 32, 32, 3)    0
```

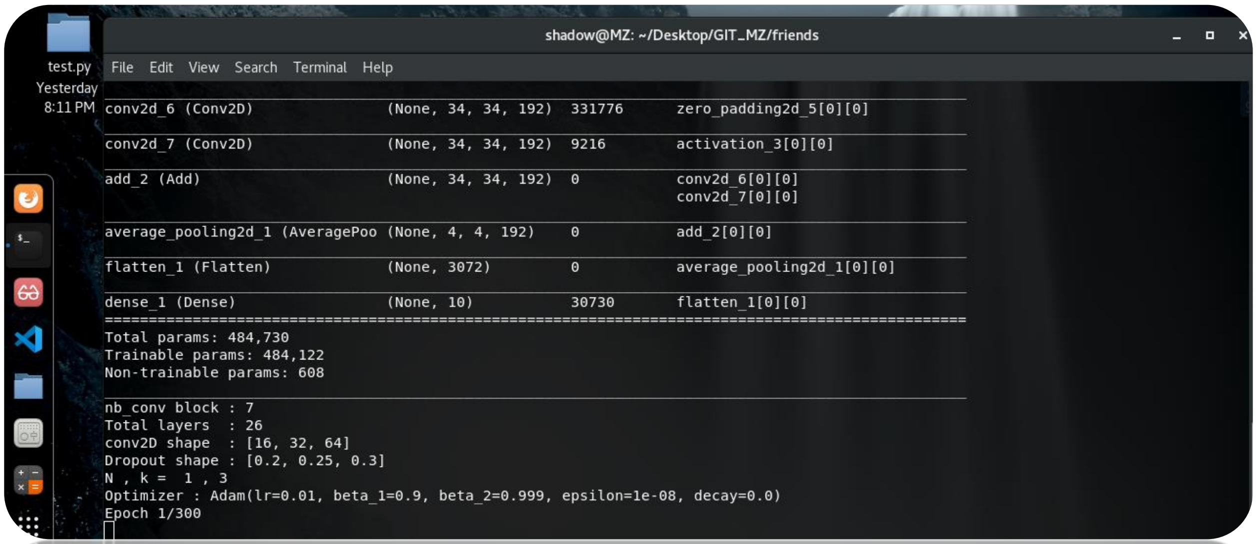
Parameters of Googlenet-resnet model



Applications
Places
Terminal
Wed Dec 4, 6:15 PM
en
test.py
Wed 8:11 PM
shadow@MZ: ~/Desktop/GIT_MZ/friends
File Edit View Search Terminal Help

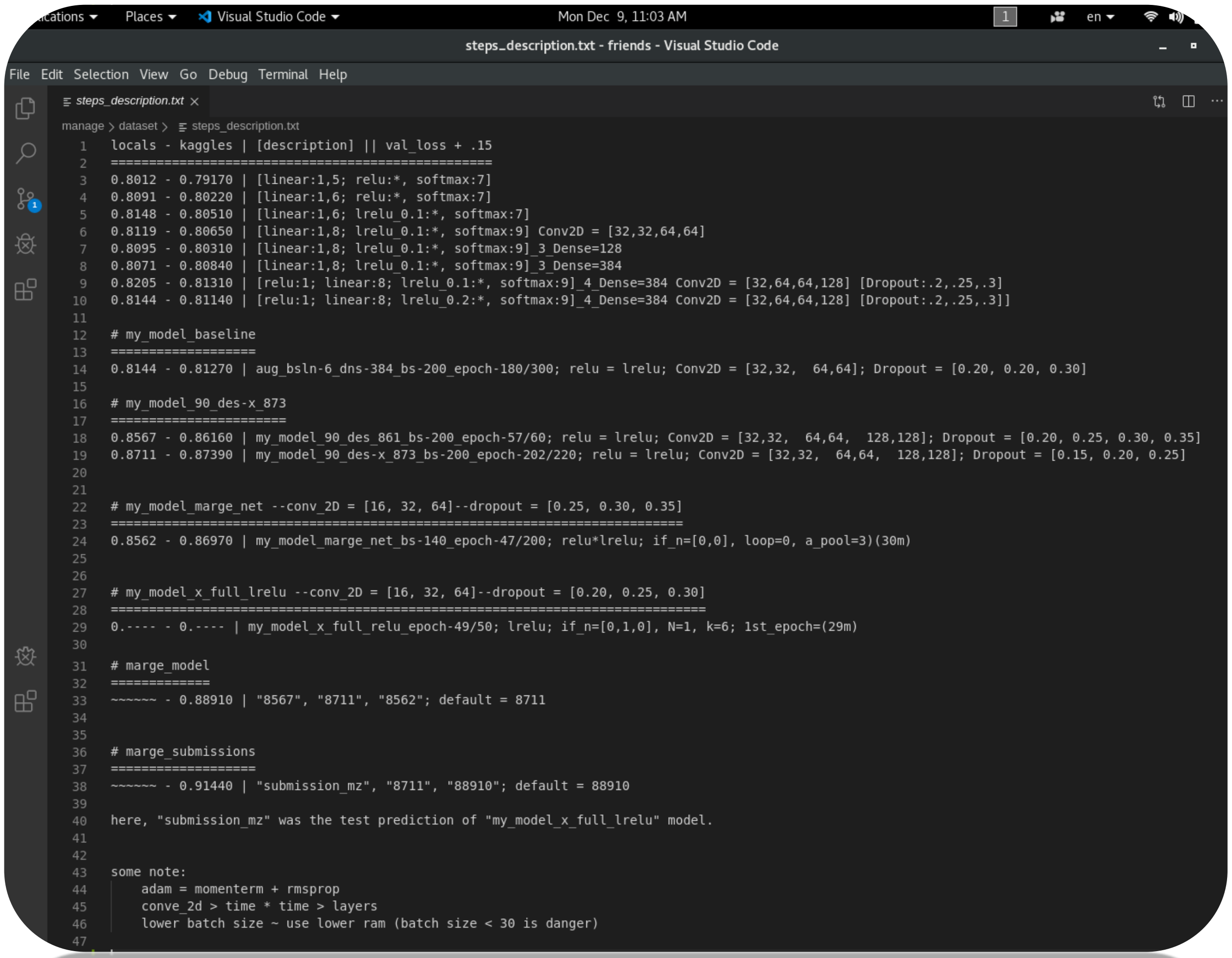
```
leaky_re_lu_1 (LeakyReLU)       (None, 32, 32, 128)  0           add_1[0][0]
____________________________________________________________________________________________
dropout_2 (Dropout)             (None, 32, 32, 128)  0           leaky_re_lu_1[0][0]
____________________________________________________________________________________________
average_pooling2d_1 (AveragePoo (None, 10, 10, 128)  0           dropout_2[0][0]
____________________________________________________________________________________________
flatten_1 (Flatten)             (None, 12800)        0           average_pooling2d_1[0][0]
____________________________________________________________________________________________
dense_1 (Dense)                 (None, 10)           128010      flatten_1[0][0]
============================================================================================
Total params: 380,554
Trainable params: 380,042
Non-trainable params: 512
____________________________________________________________________________________________
conv_n block  : 8
Total layers  : 23
loop, a_pool  : 1 , 3
conv2D shape  : [16, 32, 64]
Dropout shape : [0.25, 0.3, 0.35]
GNet          : (n, n)
Optimizer : Adam(lr=0.01, beta_1=0.9, beta_2=0.999, epsilon=1e-08, decay=0.0)
Epoch 1/300
 - 1898s - loss: 4.8807 - accuracy: 0.3946 - val_loss: 1.3126 - val_accuracy: 0.4709

Epoch 00001: val_loss improved from inf to 1.31258, saving model to /home/shadow/Desktop/GIT_MZ/friends/manage/checkpoint/model.h5
```

Parameters of model-X


test.py
Yesterday
8:11 PM
shadow@MZ: ~/Desktop/GIT_MZ/friends
File Edit View Search Terminal Help

```
conv2d_6 (Conv2D)               (None, 34, 34, 192)  331776      zero_padding2d_5[0][0]
________________________________________________________________________________________________
conv2d_7 (Conv2D)               (None, 34, 34, 192)  9216        activation_3[0][0]
________________________________________________________________________________________________
add_2 (Add)                     (None, 34, 34, 192)  0           conv2d_6[0][0]
                                                                 conv2d_7[0][0]
________________________________________________________________________________________________
average_pooling2d_1 (AveragePoo (None, 4, 4, 192)    0           add_2[0][0]
________________________________________________________________________________________________
flatten_1 (Flatten)             (None, 3072)         0           average_pooling2d_1[0][0]
________________________________________________________________________________________________
dense_1 (Dense)                 (None, 10)           30730       flatten_1[0][0]
================================================================================================
Total params: 484,730
Trainable params: 484,122
Non-trainable params: 608
________________________________________________________________________________________________
nb_conv block : 7
Total layers  : 26
conv2D shape  : [16, 32, 64]
Dropout shape : [0.2, 0.25, 0.3]
N , k =  1 , 3
Optimizer : Adam(lr=0.01, beta_1=0.9, beta_2=0.999, epsilon=1e-08, decay=0.0)
Epoch 1/300
```

steps_description.txt



```
locals - kaggles | [description] || val_loss + .15
================================================
0.8012 - 0.79170 | [linear:1,5; relu:*, softmax:7]
0.8091 - 0.80220 | [linear:1,6; relu:*, softmax:7]
0.8148 - 0.80510 | [linear:1,6; lrelu_0.1:*, softmax:7]
0.8119 - 0.80650 | [linear:1,8; lrelu_0.1:*, softmax:9] Conv2D = [32,32,64,64]
0.8095 - 0.80310 | [linear:1,8; lrelu_0.1:*, softmax:9]_3_Dense=128
0.8071 - 0.80840 | [linear:1,8; lrelu_0.1:*, softmax:9]_3_Dense=384
0.8205 - 0.81310 | [relu:1; linear:8; lrelu_0.1:*, softmax:9]_4_Dense=384 Conv2D = [32,64,64,128] [Dropout:.2,.25,.3]
0.8144 - 0.81140 | [relu:1; linear:8; lrelu_0.2:*, softmax:9]_4_Dense=384 Conv2D = [32,64,64,128] [Dropout:.2,.25,.3]]

# my_model_baseline
===================
0.8144 - 0.81270 | aug_bsln-6_dns-384_bs-200_epoch-180/300; relu = lrelu; Conv2D = [32,32,  64,64]; Dropout = [0.20, 0.20, 0.30]

# my_model_90_des-x_873
=======================
0.8567 - 0.86160 | my_model_90_des_861_bs-200_epoch-57/60; relu = lrelu; Conv2D = [32,32,  64,64,  128,128]; Dropout = [0.20, 0.25, 0.30, 0.35]
0.8711 - 0.87390 | my_model_90_des-x_873_bs-200_epoch-202/220; relu = lrelu; Conv2D = [32,32,  64,64,  128,128]; Dropout = [0.15, 0.20, 0.25]


# my_model_marge_net --conv_2D = [16, 32, 64]--dropout = [0.25, 0.30, 0.35]
==========================================================================
0.8562 - 0.86970 | my_model_marge_net_bs-140_epoch-47/200; relu*lrelu; if_n=[0,0], loop=0, a_pool=3)(30m)


# my_model_x_full_lrelu --conv_2D = [16, 32, 64]--dropout = [0.20, 0.25, 0.30]
=============================================================================
0.---- - 0.---- | my_model_x_full_relu_epoch-49/50; lrelu; if_n=[0,1,0], N=1, k=6; 1st_epoch=(29m)

# marge_model
=============
~~~~~~ - 0.88910 | "8567", "8711", "8562"; default = 8711


# marge_submissions
===================
~~~~~~ - 0.91440 | "submission_mz", "8711", "88910"; default = 88910

here, "submission_mz" was the test prediction of "my_model_x_full_lrelu" model.


some note:
    adam = momenterm + rmsprop
    conve_2d > time * time > layers
    lower batch size ~ use lower ram (batch size < 30 is danger)
```

Conclusion:

We were trying to do our best. But for the time limitation and also for the big challenge of running the models in our local computer we couldn't present our best. As like we says in "**steps_ description.txt**":

1. Using larger convolution layers reduces the speed of training with a big rate.
2. Even using larger filter in convolution reduces speed of training but not as 1.
3. Higher batch size gives better validation accuracy and less than 30 is danger.

4. But using higher batch size makes higher ram use. Even for it we have faced unwanted termination of training in almost time. We even have got 7 termination while using "Googlenet-resnet merge model" in sequence after running every 1.5-2 hours. It's because deeper epochs need more ram than the beginner epochs. We then have used 200 batch size and even get caught at 160 Batch size. Our PC always has kept free while training (In Linux only).

But after all those have done, we have earned our success. I think it's enough for us as beginner. And we have achieved a much knowledge and patience through this project. A big thanks to our teacher who gives us this opportunity to introduce ourselves with the recent world and the heart of it. I think we should implement these achievements in our lives so that we can clarify our position in this race of creativity.