Homework3

Classify ‘Cat’ and ‘Dog’ Pictures using 5-model

“(1) Simple CNN, (2) AlexNet, (3) GoogleInceptionNet,
(4) VGGNet, (5) Residual Network”

Dept. of CSE, Yonsei University
2016323246 Junyoung Jang

In this report, I used Tensorflow Library which easily use Convolution Neural Network to train given data set. (I agree with that it is necessary manually to make the algorithm and code, but we can expect more exact result by using Tensorflow Library.) This report introduce each steps that are involved to Computing Environment, Model Setting, Test results.

- Computing Environment

Unlike MATLAB, Tensorflow is sensitive to computing environment. If you want to run my code, you should set the same environment at least.

- OS : Windows
  (if you use Linux or OSX, you must replace string of load directory '\\' to '/')
- Python version : 3.6.0
- IDE : Pycharm Community 2016.3
- Library : Anaconda, Tensorflow 1.0.0 (+tflearn library)

- Model Setting

It used given data and another pictures (Cat & Dog) that I downloaded "Cat & Dog" Picture as "Kaggle(http://www.kaggle.com)" but **I did not use test1.zip (#25,000pic) file**. Because I think it is data snooping(cheating). Therefore, I used given test data(#2,000pic) + extra data(#1,000pic) + Kaggle Train data(#24,000pic) to train and I only used given test picture(#600pic) for validation. I think overfitting occur due to insufficient amount of data (amount of data is smaller than MNIST data!). Therefore, I construct 5-models to evaluate "Cat & Dog" pictures. In the report, used models is as below:

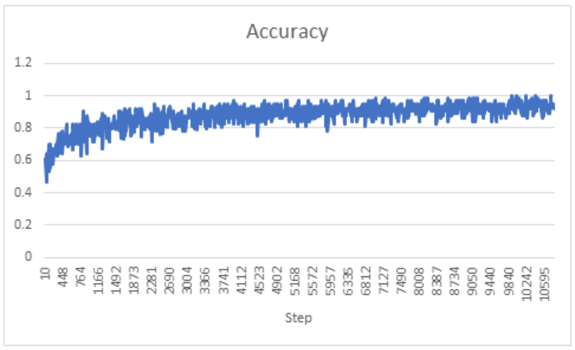
"(1) Simple CNN, (2) AlexNet, (3) GoogleInceptionNet,
(4) VGGNet, (5) Residual Network"

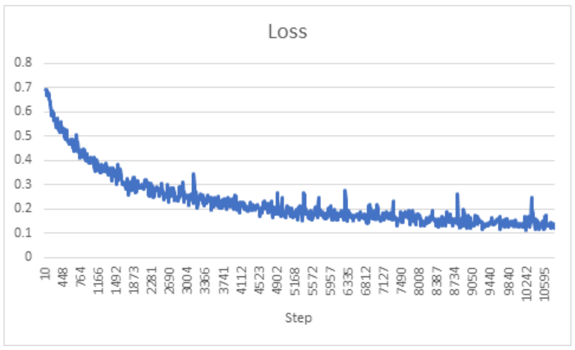
I classify pictures as like election("다수결") using 5-model. For example, if I have "classified picture – Cat" using 5-model, it mean that 5-model have result as [(1), (2), (3), (4), (5)] = [0, 0, 0, 0, 1] or [0, 0, 0, 1, 1] etc. (0:Cat, 1: Dog). To train model, I set up the models in common environment as below:

- Trained #Pictures : 27,000, Cat(12,980) + Dog(14,020)
- Validation #Pictures : 600, Cat(298) + Dog(302)
- Image resize : 64×64×3
- Training Epoch : 100
- Learning Rate : 0.0005
- Optimization method : Adam, RMSProp(VGGNet), Momentum(ResNet)
- Batch Size : 64
- Drop rate : 50%

Also, I used training techniques which are normalization of image and Flipping & Rotating & Cropping (Synthetic training data). I explain flow of each algorithm using Tensorboard through “Appendix 2” (“Appendix 1” include Code of each algorithm). Next, I'll briefly explain Accuracy and Loss values of each method. And I finally explain how to use "test.py" to run code and to evaluate picture.

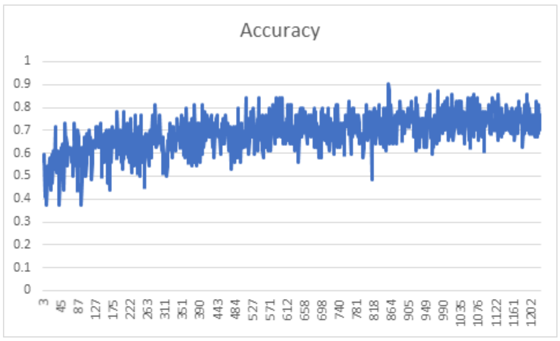
1-Method : Simple CNN

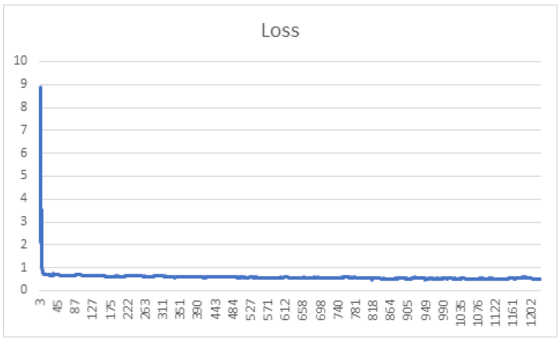




I simulated the model running #epoch 100. As result, I have got the accuracy about 95% and the Loss value about 0.12.

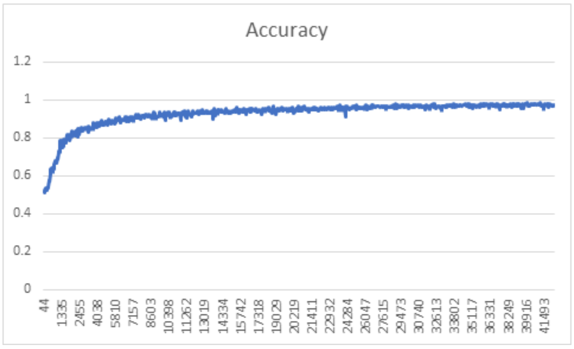
2-Method : AlexNet

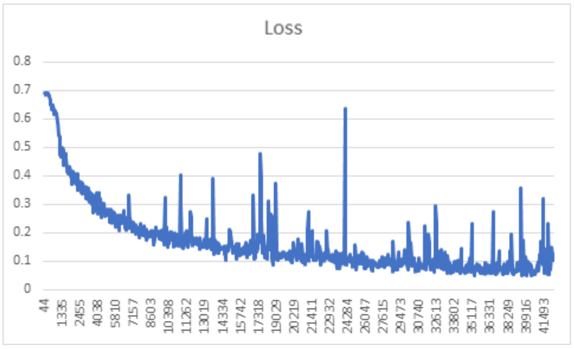




I simulated the model running #epoch 100. As result, I have got the accuracy about 83% and the Loss value about 0.5.

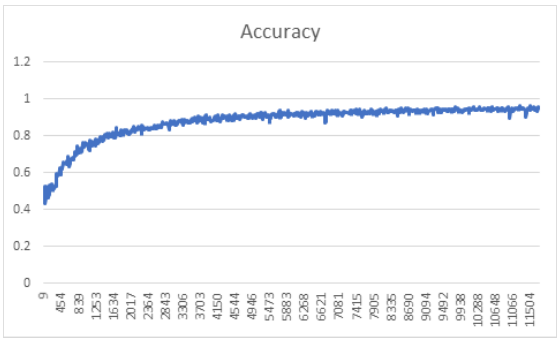
3-Method : GoogleInceptionNet

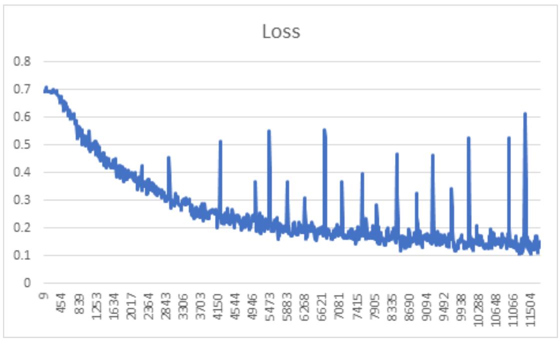




I simulated the model running #epoch 100. As result, I have got the accuracy about 97% and the Loss value about 0.12.

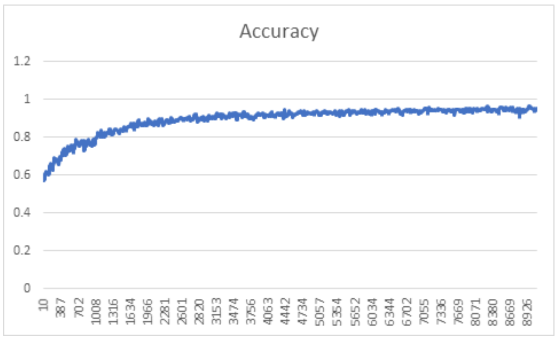
4-Method : VGGNet

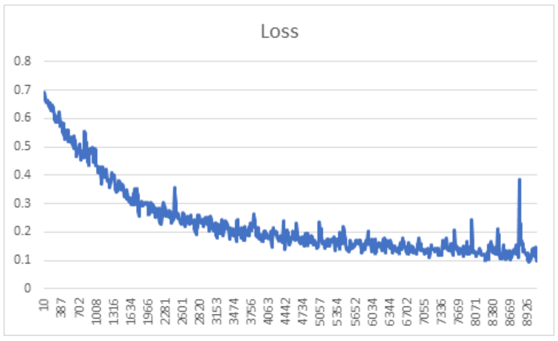




I simulated the model running #epoch 100. As result, I have got the accuracy about 97% and the Loss value about 0.1.

5-Method : Residual Network



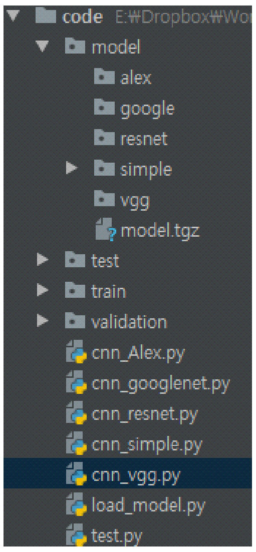


I simulated the model running #epoch 100. As result, I have got the accuracy about 95% and the Loss value about 0.11.

Finally, I introduce

How to use "test.py"

[Download Model]



I set automatically download 5-model in "test.py" code usign my FTP server. If you can not download or take place error, you can download (about 700MB) below address:

Dropbox :https://www.dropbox.com/s/cxu1qz0ui5dy56p/model.tgz?dl=0

FTP :http://junyoungjang.ipdisk.co.kr:20925/publist/HDD1/EveryoneShare/model.tgz

Also, You must extract the *.tgz file in present working directory like the right pictures.

[Download Trained Set and New Training Model]

If you want to download Trained Set(and Validation Set) as below address:

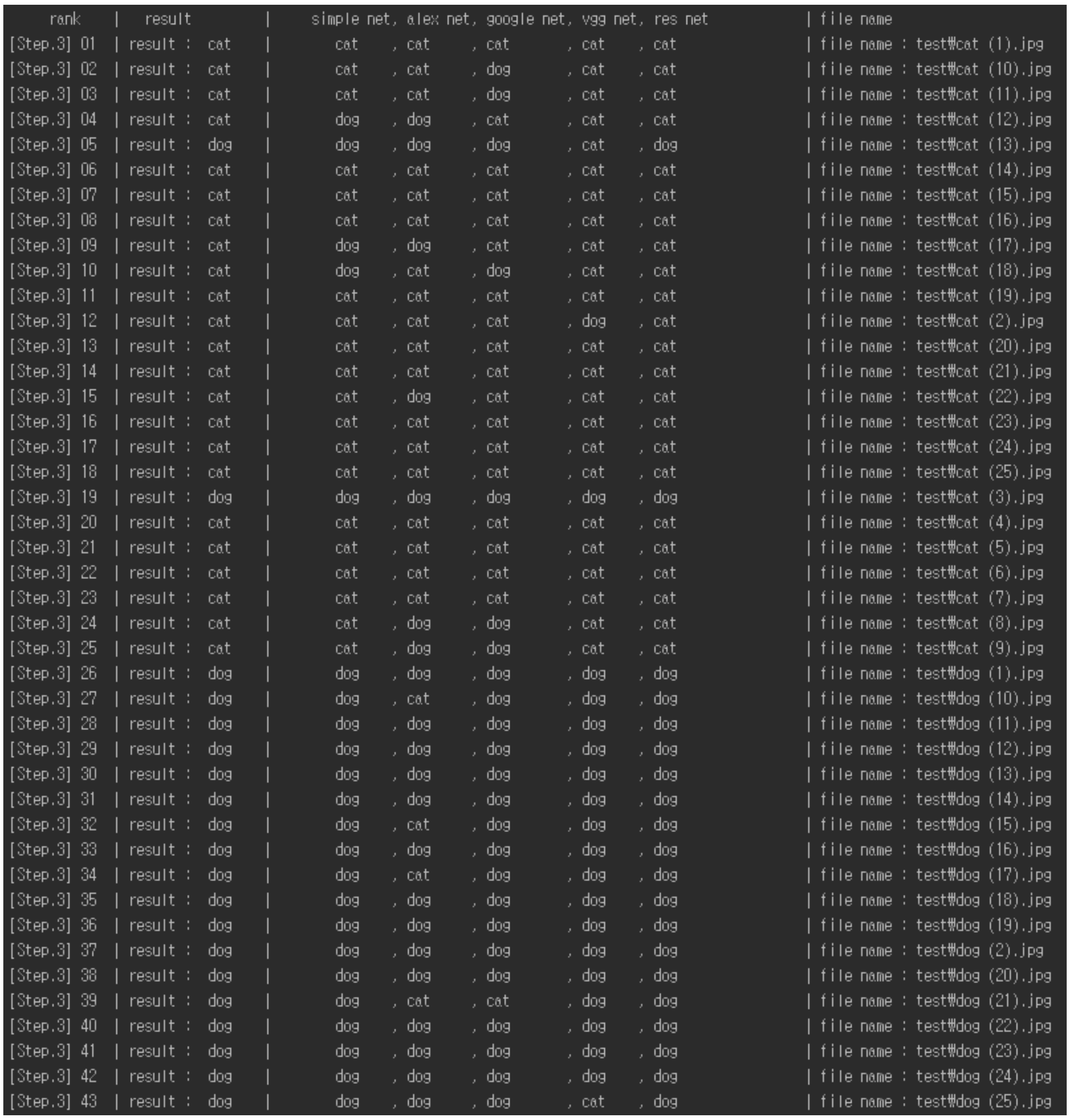
FTP :http://junyoungjang.ipdisk.co.kr:20925/publist/HDD1/EveryoneShare/train_val_set.tgz

It is heavy volume (about 600MB). Also, If you want to train model, you run the code (model_simple.py, model_Alex.py, etc)

[Test]

If you want to test 5-model by running 50 pictures, you just put image to "test" folder. I recommend using "jpg" or "jpeg" files. And I print result out as below: (If you want to only see result, you just see "result column")

rank	result	methods	file name
1	**cat**	cat, cat, cat, dog, dog	file name : test.jpg
…	**…**	…	…
50	**dog**	dog, cat, cat, dog, dog	file name : test_final.jpg



```
    rank     |  result        |     simple net, alex net, google net, vgg net, res net           | file name
[Step.3] 01  | result :  cat  |        cat    , cat     , cat        , cat    , cat              | file name : test#cat (1).jpg
[Step.3] 02  | result :  cat  |        cat    , cat     , dog        , cat    , cat              | file name : test#cat (10).jpg
[Step.3] 03  | result :  cat  |        cat    , cat     , dog        , cat    , cat              | file name : test#cat (11).jpg
[Step.3] 04  | result :  cat  |        dog    , dog     , cat        , cat    , cat              | file name : test#cat (12).jpg
[Step.3] 05  | result :  dog  |        dog    , dog     , dog        , cat    , dog              | file name : test#cat (13).jpg
[Step.3] 06  | result :  cat  |        cat    , cat     , cat        , cat    , cat              | file name : test#cat (14).jpg
[Step.3] 07  | result :  cat  |        cat    , cat     , cat        , cat    , cat              | file name : test#cat (15).jpg
[Step.3] 08  | result :  cat  |        cat    , cat     , cat        , cat    , cat              | file name : test#cat (16).jpg
[Step.3] 09  | result :  cat  |        dog    , dog     , cat        , cat    , cat              | file name : test#cat (17).jpg
[Step.3] 10  | result :  cat  |        dog    , cat     , dog        , cat    , cat              | file name : test#cat (18).jpg
[Step.3] 11  | result :  cat  |        cat    , cat     , cat        , cat    , cat              | file name : test#cat (19).jpg
[Step.3] 12  | result :  cat  |        cat    , cat     , cat        , dog    , cat              | file name : test#cat (2).jpg
[Step.3] 13  | result :  cat  |        cat    , cat     , cat        , cat    , cat              | file name : test#cat (20).jpg
[Step.3] 14  | result :  cat  |        cat    , cat     , cat        , cat    , cat              | file name : test#cat (21).jpg
[Step.3] 15  | result :  cat  |        cat    , dog     , cat        , cat    , cat              | file name : test#cat (22).jpg
[Step.3] 16  | result :  cat  |        cat    , cat     , cat        , cat    , cat              | file name : test#cat (23).jpg
[Step.3] 17  | result :  cat  |        cat    , cat     , cat        , cat    , cat              | file name : test#cat (24).jpg
[Step.3] 18  | result :  cat  |        cat    , cat     , cat        , cat    , cat              | file name : test#cat (25).jpg
[Step.3] 19  | result :  dog  |        dog    , dog     , dog        , dog    , dog              | file name : test#cat (3).jpg
[Step.3] 20  | result :  cat  |        cat    , cat     , cat        , cat    , cat              | file name : test#cat (4).jpg
[Step.3] 21  | result :  cat  |        cat    , cat     , cat        , cat    , cat              | file name : test#cat (5).jpg
[Step.3] 22  | result :  cat  |        cat    , cat     , cat        , cat    , cat              | file name : test#cat (6).jpg
[Step.3] 23  | result :  cat  |        cat    , cat     , cat        , cat    , cat              | file name : test#cat (7).jpg
[Step.3] 24  | result :  cat  |        cat    , dog     , dog        , cat    , cat              | file name : test#cat (8).jpg
[Step.3] 25  | result :  cat  |        cat    , dog     , dog        , cat    , cat              | file name : test#cat (9).jpg
[Step.3] 26  | result :  dog  |        dog    , dog     , dog        , dog    , dog              | file name : test#dog (1).jpg
[Step.3] 27  | result :  dog  |        dog    , cat     , dog        , dog    , dog              | file name : test#dog (10).jpg
[Step.3] 28  | result :  dog  |        dog    , dog     , dog        , dog    , dog              | file name : test#dog (11).jpg
[Step.3] 29  | result :  dog  |        dog    , dog     , dog        , dog    , dog              | file name : test#dog (12).jpg
[Step.3] 30  | result :  dog  |        dog    , dog     , dog        , dog    , dog              | file name : test#dog (13).jpg
[Step.3] 31  | result :  dog  |        dog    , dog     , dog        , dog    , dog              | file name : test#dog (14).jpg
[Step.3] 32  | result :  dog  |        dog    , cat     , dog        , dog    , dog              | file name : test#dog (15).jpg
[Step.3] 33  | result :  dog  |        dog    , dog     , dog        , dog    , dog              | file name : test#dog (16).jpg
[Step.3] 34  | result :  dog  |        dog    , cat     , dog        , dog    , dog              | file name : test#dog (17).jpg
[Step.3] 35  | result :  dog  |        dog    , dog     , dog        , dog    , dog              | file name : test#dog (18).jpg
[Step.3] 36  | result :  dog  |        dog    , dog     , dog        , dog    , dog              | file name : test#dog (19).jpg
[Step.3] 37  | result :  dog  |        dog    , dog     , dog        , dog    , dog              | file name : test#dog (2).jpg
[Step.3] 38  | result :  dog  |        dog    , dog     , dog        , dog    , dog              | file name : test#dog (20).jpg
[Step.3] 39  | result :  dog  |        dog    , cat     , cat        , dog    , dog              | file name : test#dog (21).jpg
[Step.3] 40  | result :  dog  |        dog    , dog     , dog        , dog    , dog              | file name : test#dog (22).jpg
[Step.3] 41  | result :  dog  |        dog    , dog     , dog        , dog    , dog              | file name : test#dog (23).jpg
[Step.3] 42  | result :  dog  |        dog    , dog     , dog        , dog    , dog              | file name : test#dog (24).jpg
[Step.3] 43  | result :  dog  |        dog    , dog     , dog        , cat    , dog              | file name : test#dog (25).jpg
```

3 column means it represent result of each method(5-method). And 2 column represent a lot of frequencies in results of 3 column like election. Finally, in Step4, it is calculated by [result(cat picture)/total picture]*100.

[Appendix 1] Python Code

1. test.py

Step1. Load Library

```
from skimage import io
from scipy.misc import imresize
import numpy as np
import matplotlib.pyplot as plt
import os, tarfile, sys, urllib
from glob import glob
```

Step2. Automatically download 5-model

```
DATA_URL                                                                         =
'http://junyoungjang.ipdisk.co.kr:20925/publist/HDD1/EveryoneShare/model
.tgz'
dest_directory = '.\\model'
if not os.path.exists(dest_directory):
    os.makedirs(dest_directory)
filename = DATA_URL.split('/')[-1]
filepath = os.path.join(dest_directory, filename)
if not os.path.exists(filepath):
    def _progress(count, block_size, total_size):
        sys.stdout.write('\r>> Downloading %s %.1f%%' % (
            filename, float(count * block_size) / float(total_size) * 100.0))
        sys.stdout.flush()
    filepath, _ = urllib.request.urlretrieve(DATA_URL, filepath, _progress)
    print()
    statinfo = os.stat(filepath)
    print('Succesfully downloaded', filename, statinfo.st_size, 'bytes.')
    tarfile.open(filepath, 'r:gz').extractall(dest_directory)  # tarfile extract
```

Step3. Evaluating pictures function.

```
def cat_dog(result):
    index = result.index(max(result))
    prob = max(result)
    if index == 0:
        str = 'cat'
    elif index == 1:
        str = 'dog'
    return str, index, prob


def election_cat_dog(result):
    if result < 3:
        str = 'cat'
        index = 1
    else:
        str = 'dog'
        index = 0
    return str, index
```

Step4. Read and Plot Image (Problem 2-1, 2-2)

```
size_image = 64
test_path = 'test\\'
test_path = os.path.join(test_path, '*')
test_files = sorted(glob(test_path))
test_numfiles = len(test_files)
X = np.zeros((test_numfiles, size_image, size_image, 3), dtype='float64')
print('[Step.1] Read pictures. #', test_numfiles)
count = 0
num = 0
for f in test_files:
    try:
        img = io.imread(f)
        if num < 2 and (count+1)%10 == 0:
            plt.imshow(img)
            title_str ="[Step.2] A multiple of 10 - (%d/%d), File Name : %s
"%(count+1, test_numfiles, f)
            plt.title(title_str, loc='left')
            plt.show()
            num += 1
        new_img = imresize(img, (size_image, size_image, 3))
        X[count] = np.array(new_img)
        count += 1
    except:
        continue
```

Step5. Load 5-models and evaluate pictures

```
from load_model import fivemodel
result_storage_ind = 0
simple_model_data, alex_model_data, google_model_data, vgg_model_data,
res_model_data = fivemodel()
```

Step6. Print Cat or Dog (Problem 2-3)

```
print('\t rank      |     result \t\t|\t  simple net, alex net, google net, vgg
net, res net\t\t\t| file name')
for ind in range(test_numfiles):
    simple_str, simple_ind, simple_prob = cat_dog(simple_model_data[ind])
    alex_str, alex_ind, alex_prob = cat_dog(alex_model_data[ind])
    google_str, google_ind, google_prob = cat_dog(google_model_data[ind])
    vgg_str, vgg_ind, vgg_prob = cat_dog(vgg_model_data[ind])
    res_str, res_ind, res_prob = cat_dog(res_model_data[ind])

    result_str, result_ind = election_cat_dog(simple_ind + alex_ind +
google_ind + vgg_ind + res_ind)
    result_storage_ind += result_ind

    print('[Step.3]', "%02d"%(ind+1), ' | result : ', result_str, '\t|\t\t',
          simple_str, '\t,',
          alex_str, '\t  ,',
          google_str, '\t  ,',
          vgg_str, '    ,',
          res_str,
          '\t\t\t      | file name :', test_files[ind])
```

Step7. Print ratio of cat picture (Problem 2-4)

```
print('[Step.4] Cat ratio in total picture : ', "%.2f
%%"%(result_storage_ind/test_numfiles*100))
```

2. load_model.py

Step1. Load Library

```
import tensorflow as tf
import tflearn
from tflearn.layers.core import input_data, dropout, fully_connected
from tflearn.layers.conv import conv_2d, max_pool_2d, avg_pool_2d
from tflearn.layers.estimator import regression
from tflearn.layers.normalization import local_response_normalization
from tflearn.layers.merge_ops import merge
from tflearn.data_preprocessing import ImagePreprocessing
from tflearn.data_augmentation import ImageAugmentation
from skimage import io
from scipy.misc import imresize
import numpy as np
import os
from glob import glob
```

Step2. Read Image

```
def read_image():
    size_image = 64
    test_path = 'test\\'
    test_path = os.path.join(test_path, '*')
    test_files = sorted(glob(test_path))
    test_numfiles = len(test_files)
    X = np.zeros((test_numfiles, size_image, size_image, 3), dtype='float64')
    count = 0
    num = 0
    for f in test_files:
        try:
            img = io.imread(f)
            new_img = imresize(img, (size_image, size_image, 3))
            X[count] = np.array(new_img)
            count += 1
        except:
            continue
    return X
```

Step3. 5-model function

```
def fivemodel():
    ## 1.Simple
    tf.reset_default_graph()
    tflearn.config.init_training_mode()
    X = read_image()
    simple_result = simple_graph(X)

    ## 2.Alex
    tf.reset_default_graph()
    tflearn.config.init_training_mode()
    X = read_image()
    alex_result = alex_graph(X)


    # 3. Google
    tf.reset_default_graph()
    tflearn.config.init_training_mode()
    X = read_image()
    google_result = google_graph(X)

    # 4. VGG
    tf.reset_default_graph()
    tflearn.config.init_training_mode()
    X = read_image()
    vgg_result = vgg_graph(X)

    # 5. Residual
    tf.reset_default_graph()
    tflearn.config.init_training_mode()
    X = read_image()
    res_result = res_graph(X)

    return simple_result, alex_result, google_result, vgg_result, res_result
```

Step4. Simple CNN Graph

```
def simple_graph(X):
    img_prep = ImagePreprocessing()
    img_prep.add_featurewise_zero_center()
    img_prep.add_featurewise_stdnorm()
    img_aug = ImageAugmentation()
    img_aug.add_random_flip_leftright()
    img_aug.add_random_rotation(max_angle=25.)
    img_aug.add_random_crop([64, 64], padding=4)
    network = input_data(shape=[None, 64, 64, 3], data_preprocessing=img_prep,
data_augmentation=img_aug)
    conv = conv_2d(network, 32, 3, activation='relu')
    network = max_pool_2d(conv, 2)
    conv = conv_2d(network, 64, 3, activation='relu')
    network = max_pool_2d(conv, 2)
    conv = conv_2d(network, 64, 3, activation='relu')
    network = max_pool_2d(conv, 2)
    conv = conv_2d(network, 64, 3, activation='relu')
    network = max_pool_2d(conv, 2)
    network = fully_connected(network, 512, activation='relu')
    network = dropout(network, 0.5)
    network = fully_connected(network, 2, activation='softmax')
    network = regression(network, optimizer='adam',
                         loss='categorical_crossentropy',
                         learning_rate=0.0005)
    simple_model = tflearn.DNN(network)
    simple_model.load('model\\simple\\jun_simple_cat_dog_final.tflearn')
    simple_result = simple_model.predict(X)
    return simple_result
```

Step5. AlexNet Graph

```
def alex_graph(X):
    img_prep = ImagePreprocessing()
    img_prep.add_featurewise_zero_center()
    img_prep.add_featurewise_stdnorm()
    img_aug = ImageAugmentation()
    img_aug.add_random_flip_leftright()
    img_aug.add_random_rotation(max_angle=25.)
    img_aug.add_random_crop([64, 64], padding=4)
    network = input_data(shape=[None, 64, 64, 3], data_preprocessing=img_prep,
data_augmentation=img_aug)
    network = conv_2d(network, 64, 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, 2, activation='softmax')
    network = regression(network, optimizer='adam',
                         loss='categorical_crossentropy',
                         learning_rate=0.0005)
    alex_model = tflearn.DNN(network)
    alex_model.load('model\\alex\\jun_ALEX_cat_dog_final.tflearn')
    alex_result = alex_model.predict(X)
    return alex_result
```

Step6. GoogleNet Function

```
def google_graph(X):
    img_prep = ImagePreprocessing()
    img_prep.add_featurewise_zero_center()
    img_prep.add_featurewise_stdnorm()
    img_aug = ImageAugmentation()
    img_aug.add_random_flip_leftright()
    img_aug.add_random_rotation(max_angle=25.)
    img_aug.add_random_crop([64, 64], padding=4)
    network = input_data(shape=[None, 64, 64, 3], data_preprocessing=img_prep, data_augmentation=img_aug)
    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)
    network = fully_connected(pool5_7_7, 2, activation='softmax')

    network = regression(network, optimizer='adam',
                         loss='categorical_crossentropy',
                         learning_rate=0.0005)
    google_model = tflearn.DNN(network)
    google_model.load('model\\google\\jun_glnet_cat_dog_final.tflearn')
    google_result = google_model.predict(X)
    return google_result
```

Because the code is too long, please understanding you.

Step7. VGGNet Graph

```
def vgg_graph(X):
    tflearn.config.init_training_mode()
    tf.reset_default_graph()

    img_prep = ImagePreprocessing()
    img_prep.add_featurewise_zero_center()
    img_prep.add_featurewise_stdnorm()
    img_aug = ImageAugmentation()
    img_aug.add_random_flip_leftright()
    img_aug.add_random_rotation(max_angle=25.)
    img_aug.add_random_crop([64, 64], padding=4)

    network = input_data(shape=[None, 64, 64, 3], data_preprocessing=img_prep,
data_augmentation=img_aug)

    network = conv_2d(network, 64, 3, activation='relu')
    network = conv_2d(network, 64, 3, activation='relu')
    network = max_pool_2d(network, 2, strides=2)

    network = conv_2d(network, 128, 3, activation='relu')
    network = conv_2d(network, 128, 3, activation='relu')
    network = max_pool_2d(network, 2, strides=2)

    network = conv_2d(network, 256, 3, activation='relu')
    network = conv_2d(network, 256, 3, activation='relu')
    network = conv_2d(network, 256, 3, activation='relu')
    network = max_pool_2d(network, 2, strides=2)

    network = conv_2d(network, 512, 3, activation='relu')
    network = conv_2d(network, 512, 3, activation='relu')
    network = conv_2d(network, 512, 3, activation='relu')
    network = max_pool_2d(network, 2, strides=2)

    network = conv_2d(network, 512, 3, activation='relu')
    network = conv_2d(network, 512, 3, activation='relu')
    network = conv_2d(network, 512, 3, activation='relu')
    network = max_pool_2d(network, 2, strides=2)

    network = fully_connected(network, 4096, activation='relu')
    network = dropout(network, 0.5)
    network = fully_connected(network, 4096, activation='relu')
    network = dropout(network, 0.5)
    network = fully_connected(network, 2, activation='softmax')

    network = regression(network, optimizer='rmsprop',
                         loss='categorical_crossentropy',
                         learning_rate=0.0001)

    vgg_model = tflearn.DNN(network)
    vgg_model.load('model\\vgg\\jun_vgg_cat_dog_final.tflearn')
    vgg_result = vgg_model.predict(X)
    return vgg_result
```

Step8. Residual Network Graph

```
def res_graph(X):
    img_prep = ImagePreprocessing()
    img_prep.add_featurewise_zero_center()
    img_prep.add_featurewise_stdnorm()
    img_aug = ImageAugmentation()
    img_aug.add_random_flip_leftright()
    img_aug.add_random_rotation(max_angle=25.)
    img_aug.add_random_crop([64, 64], padding=4)

    n = 5
    net = input_data(shape=[None, 64, 64, 3], data_preprocessing=img_prep,
data_augmentation=img_aug)
    net = tflearn.conv_2d(net, 16, 3, regularizer='L2', weight_decay=0.0001)
    net = tflearn.residual_block(net, n, 16)
    net = tflearn.residual_block(net, 1, 32, downsample=True)
    net = tflearn.residual_block(net, n - 1, 32)
    net = tflearn.residual_block(net, 1, 64, downsample=True)
    net = tflearn.residual_block(net, n - 1, 64)
    net = tflearn.batch_normalization(net)
    net = tflearn.activation(net, 'relu')
    net = tflearn.global_avg_pool(net)
    # Regression
    net = tflearn.fully_connected(net, 2, activation='softmax')
    mom = tflearn.Momentum(0.1, lr_decay=0.1, decay_step=32000, staircase=True)
    net = tflearn.regression(net, optimizer=mom,
                             loss='categorical_crossentropy')
    model = tflearn.DNN(net)
    # rnn typo -> res
    model.load('model\\res\\jun_rnn_cat_dog.tflearn')
    res_result = model.predict(X)
    return res_result
```

3. Making Model Code
- Common part

Step1. Import Library

```
from __future__ import division, print_function, absolute_import
from skimage import color, io
from scipy.misc import imresize
import numpy as np
import os
from glob import glob
import tflearn
from tflearn.data_utils import shuffle, to_categorical
from tflearn.layers.core import input_data, dropout, fully_connected
from tflearn.layers.conv import conv_2d, max_pool_2d, avg_pool_2d
from tflearn.layers.estimator import regression
from tflearn.data_preprocessing import ImagePreprocessing
from tflearn.data_augmentation import ImageAugmentation
from tflearn.metrics import Accuracy
```

Step2. Import Test Picture

```
train_path = 'train\\'
train_cat_path = os.path.join(train_path, 'cat\\*.jpg')
train_dog_path = os.path.join(train_path, 'dog\\*.jpg')
train_cat_files = sorted(glob(train_cat_path))
train_dog_files = sorted(glob(train_dog_path))
train_numfiles = len(train_cat_files) + len(train_dog_files)
print(' - Number of train set : ', train_numfiles)
size_image = 64
trainX = np.zeros((train_numfiles, size_image, size_image, 3), dtype='float64')
trainY = np.zeros(train_numfiles)
count = 0
for f in train_cat_files:
    try:
        img = io.imread(f)
        new_img = imresize(img, (size_image, size_image, 3))
        trainX[count] = np.array(new_img)
        trainY[count] = 0
        count += 1
    except:
        continue

for f in train_dog_files:
    try:
        img = io.imread(f)
        new_img = imresize(img, (size_image, size_image, 3))
        trainX[count] = np.array(new_img)
        trainY[count] = 1
        count += 1
    except:
        continue
```

Step3. Import Validation Picture

```
test_path = 'validation\\'
test_cat_path = os.path.join(test_path, 'cat\\*.jpg')
test_dog_path = os.path.join(test_path, 'dog\\*.jpg')
test_cat_files = sorted(glob(test_cat_path))
test_dog_files = sorted(glob(test_dog_path))
test_numfiles = len(test_cat_files) + len(test_dog_files)
print(' - Number of test set : ', test_numfiles)
testX = np.zeros((test_numfiles, size_image, size_image, 3), dtype='float64')
testY = np.zeros(test_numfiles)
count = 0
for f in test_cat_files:
    try:
        img = io.imread(f)
        new_img = imresize(img, (size_image, size_image, 3))
        testX[count] = np.array(new_img)
        testY[count] = 0
        count += 1
    except:
        continue

for f in test_dog_files:
    try:
        img = io.imread(f)
        new_img = imresize(img, (size_image, size_image, 3))
        testX[count] = np.array(new_img)
        testY[count] = 1
        count += 1
    except:
        continue
```

Step4. Image transformation

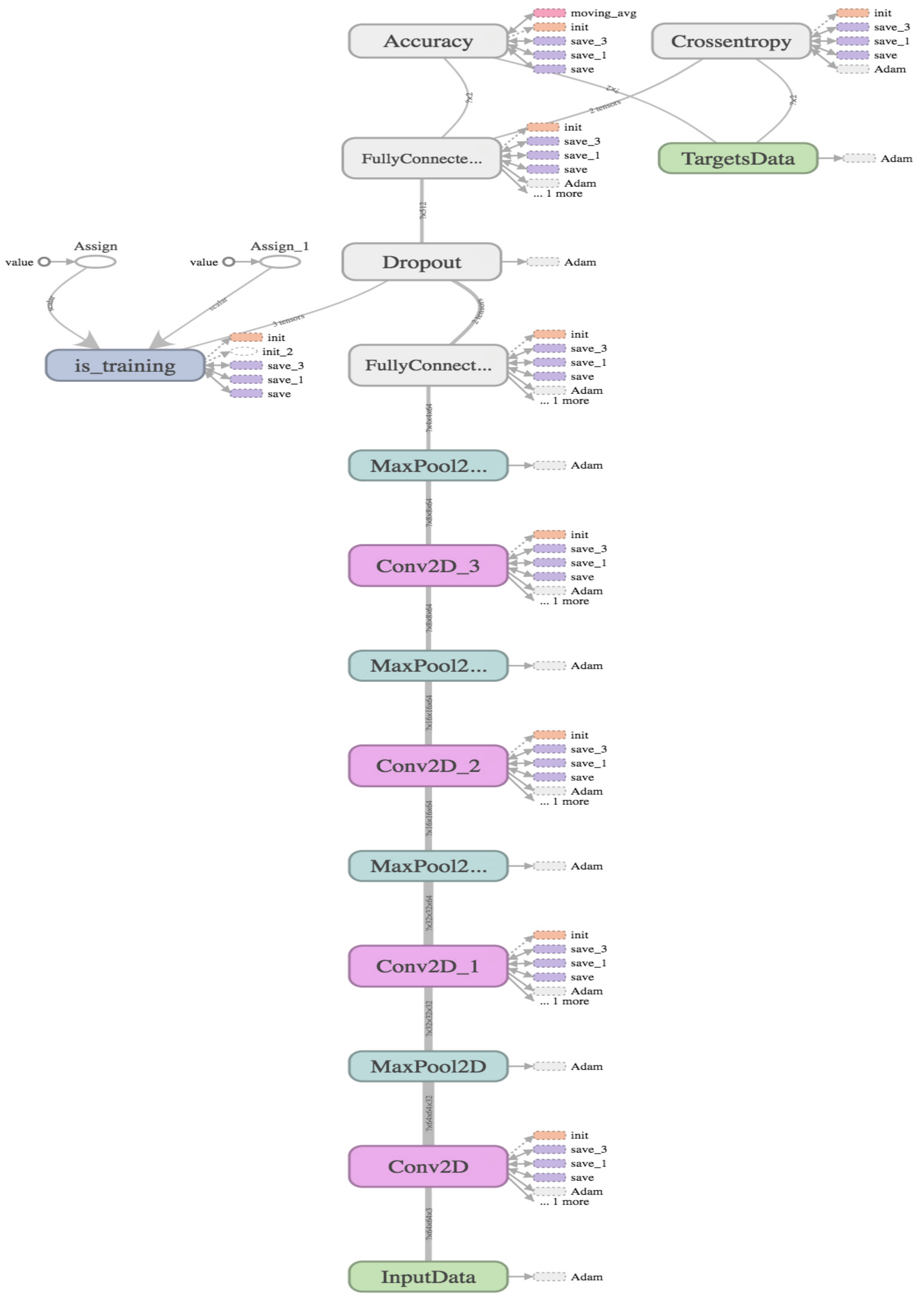
```
# Normalization of images
img_prep = ImagePreprocessing()
img_prep.add_featurewise_zero_center()
img_prep.add_featurewise_stdnorm()

# Create extra synthetic training data by flipping & rotating images
img_aug = ImageAugmentation()
img_aug.add_random_flip_leftright()
img_aug.add_random_rotation(max_angle=25.)
img_aug.add_random_crop([64, 64], padding=4)
```

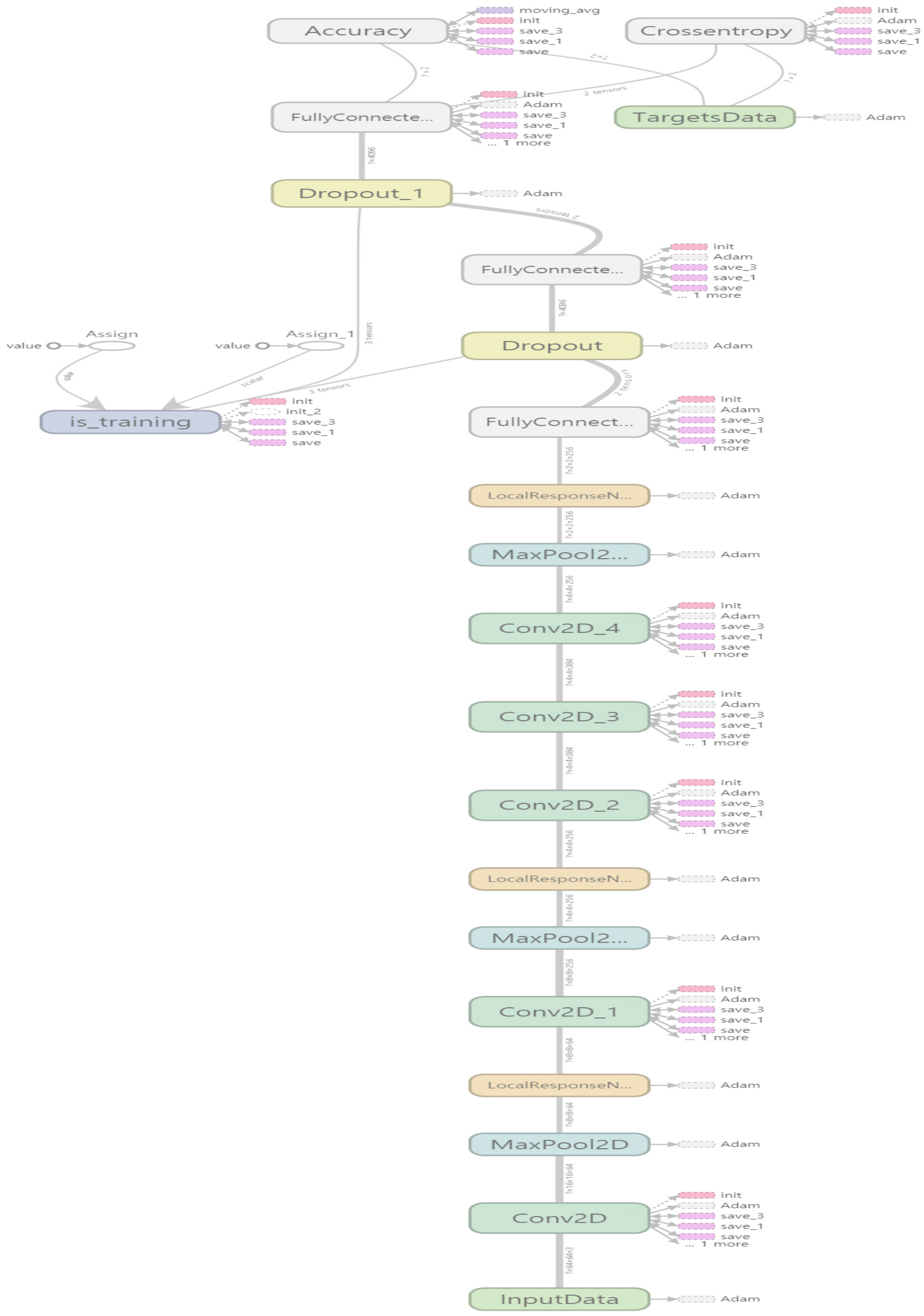
The graph(Algorithm) of Each method is equal to of Step4 - Step 8 of ‘load_model.py’

[Appendix 2] Flow of Each Algorithm

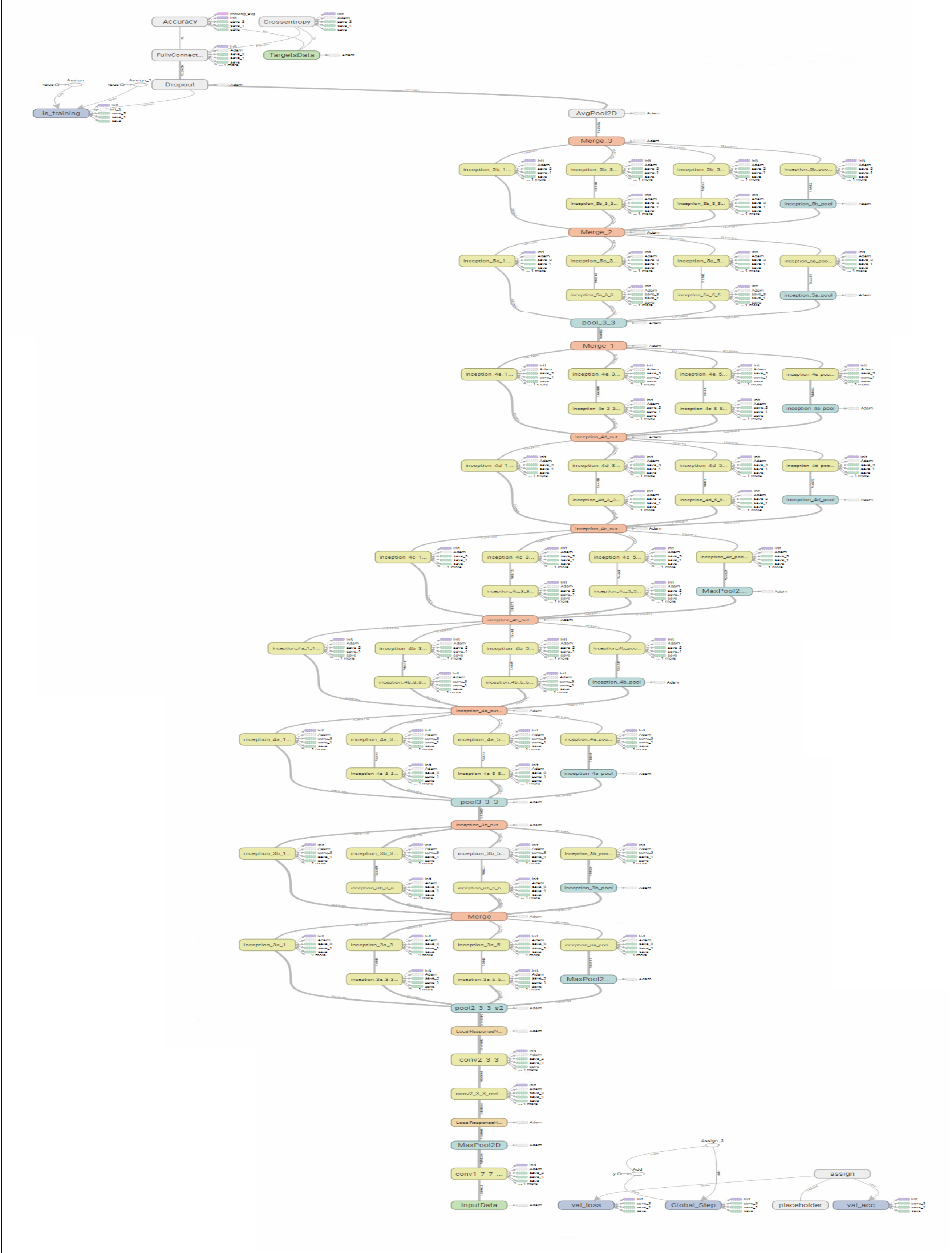
1-Model. Simple CNN



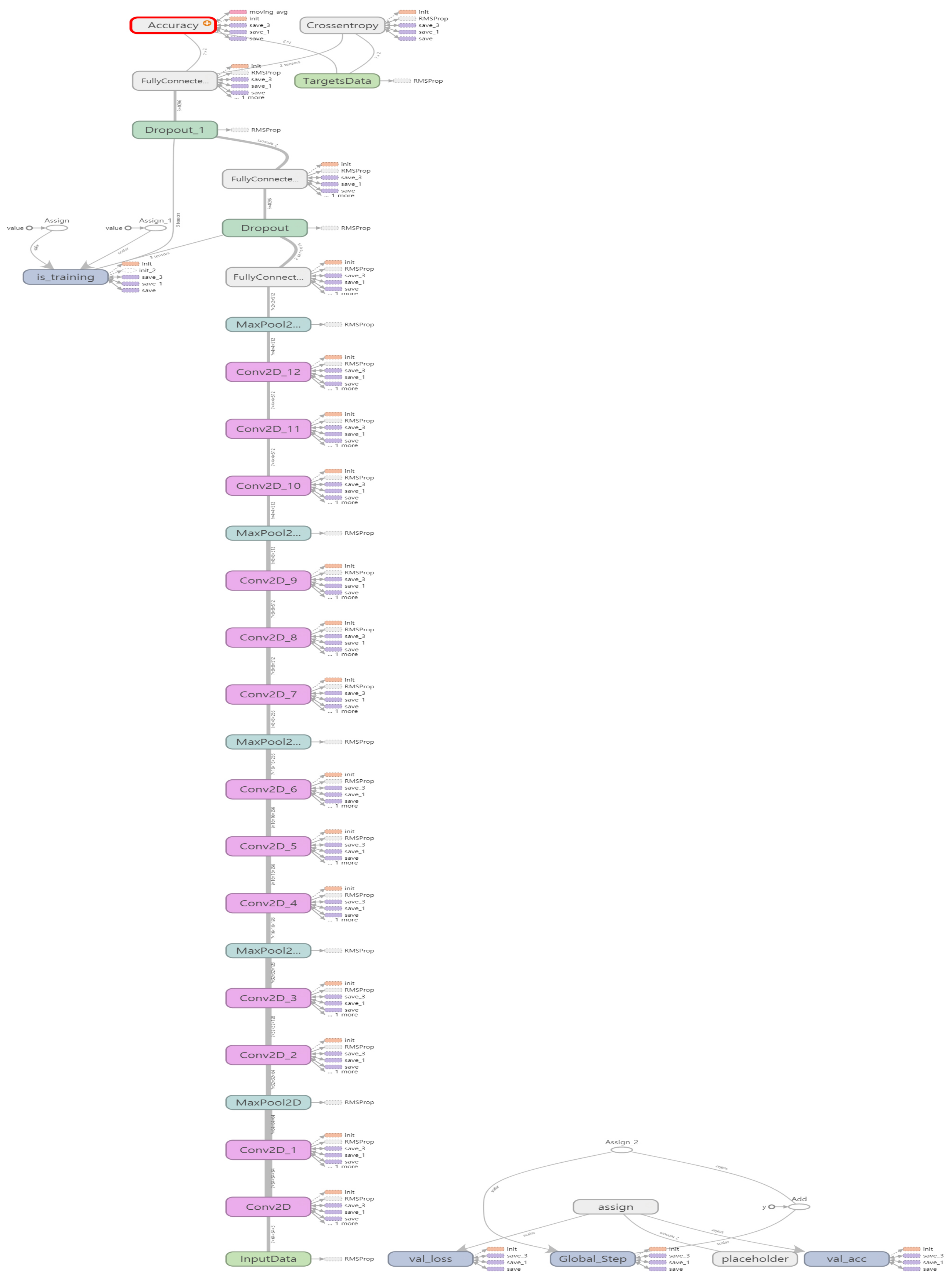
2-Model. AlexNet



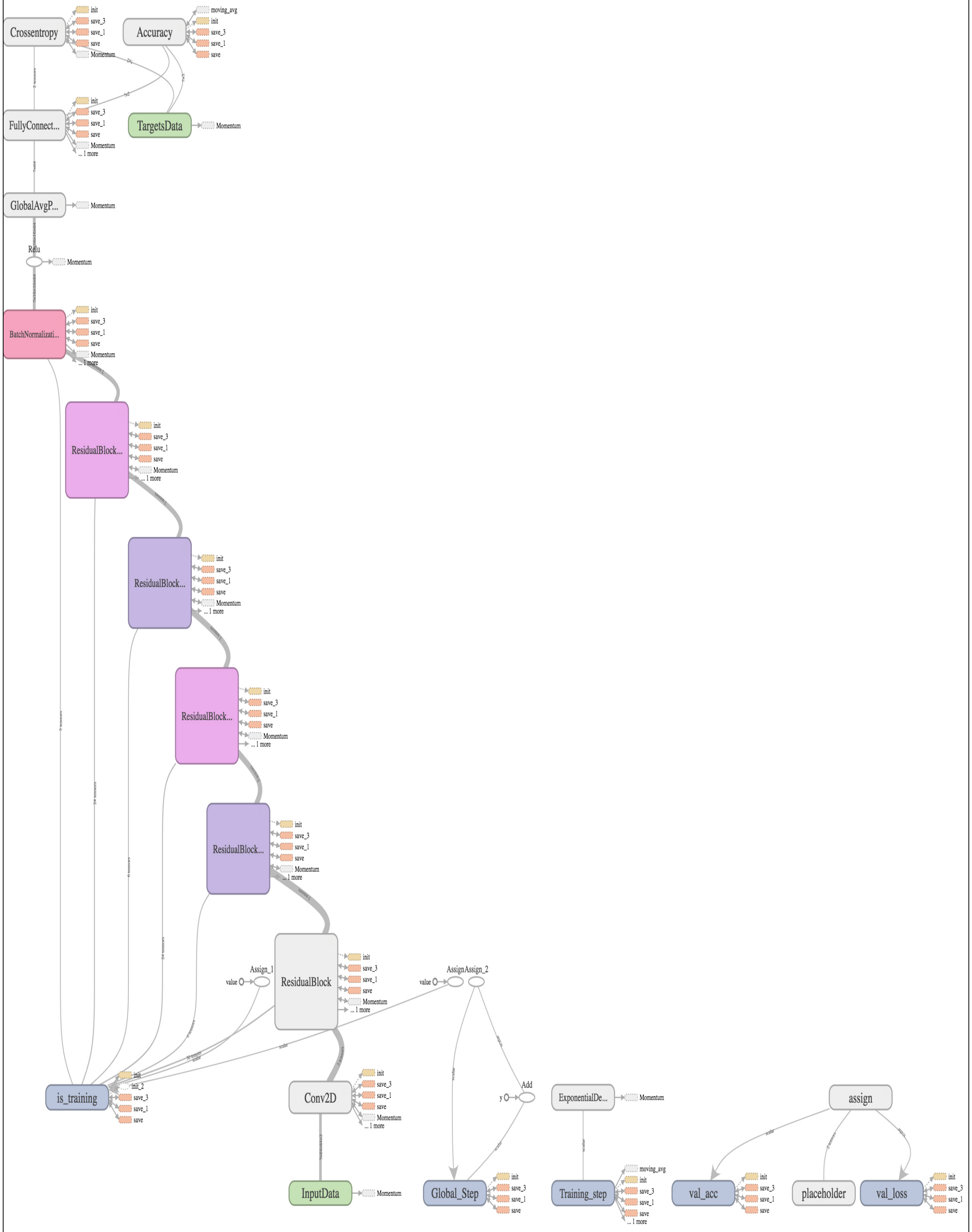
3-Model. GoogleNet



4-Model. VGGNet



5-Model. Residual Network



Crossentropy
Accuracy
FullyConnect...
TargetsData
GlobalAvgP...
Relu
BatchNormalizati...
ResidualBlock...
ResidualBlock...
ResidualBlock...
ResidualBlock...
ResidualBlock
Assign_1
AssignAssign_2
Add
is_training
Conv2D
InputData
Global_Step
ExponentialDe...
Training_step
assign
val_acc
placeholder
val_loss
init
save_3
save_1
save
Momentum
moving_avg
... 1 more