Deep Learning Project

Face Detection and Face recognition with Caffe

12/05/2017

Introduction

Face detection and recognition are both widely discussed topics in the field of deep learning or in a larger computer vision. Face detection and recognition systems can quickly and accurately identify target individuals when the conditions are favorable with facial landmark, name, gender, emotion and other related identification. Actually, Face detection and recognition have been lived with us for a such long time even we don't realize. Starting from the old days, Facebook used to make us to tag our friends in photos by clicking on them and typing in their names. Now as soon as we upload a photo, Facebook tags everyone for us like magic.

Specifically to our final project of Machine Learning II class, the main functionalities is by creating a Graphic User Interface window, it is possible to read video from file or webcam and play video from that file or webcam, do face detection from video frames and face alignment from that frames, display face detection and recognition results including names, facial landmarks and genders. This project includes but is not restricted to Caffe, OpenCV, Python, Qt and Dlib for final displaying. Here is a general project flow combining with frameworks we used for reference:

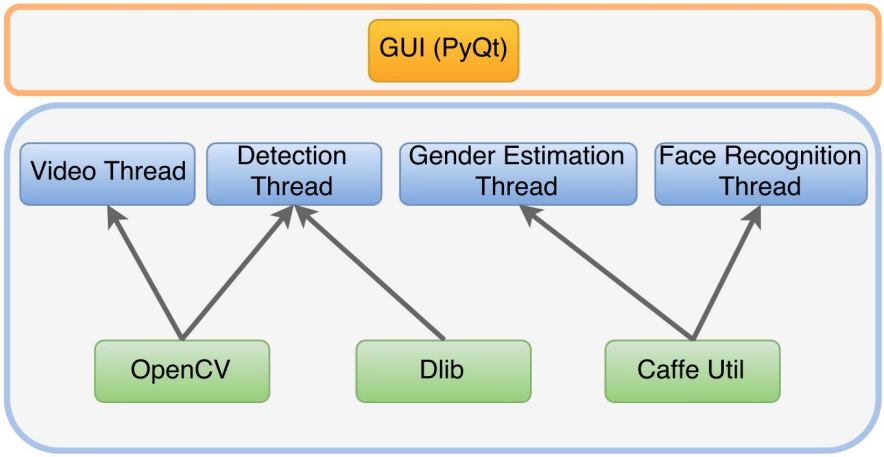


Figure: project flow

Description of the Data Set

The database we used for face recognition is Labeled Faces in the Wild (LFW) from University of Massachusetts. It is a database of face photographs designed for studying the problem of unconstrained face recognition and contains more than 13,000 images of faces collected from the web. Each face has been labeled with the name of the person pictured. 1680 of all people pictured have two or more distinct photos in the database. The only constraint on these faces is that they were detected by the Viola-Jones face detector.

LFW database contains 13,233 images for 5,749 people. Each data is labeled, we may use the data set to train deep network. Since caffe models always are pre-trained by professional groups or long term researches, we utilized LFW database for finetuning only based on original caffe model. In this case, we grouped a subset from LFW database with people who have more than 20 pictures identification. As a result, we have a database of 62 people including 2392 pictures of training set and 631 pictures of testing set.

Description of the deep learning network and training algorithm

At this point, AlexNet, VGG, GoogleNet, and ResNet are the four being most acceptable to public. These networks represent the epochs of development on deep learning world. AlexNet was the earliest network making neuron network significantly powerful. AlexNet contained only 8 layers, first 5 were convolutional layers followed by fully connected layers. Although it has been almost deprecated along the fast changes on neuron network field. GoogleNet and ResNet are also two most powerful networks that are widely used in industrial field.

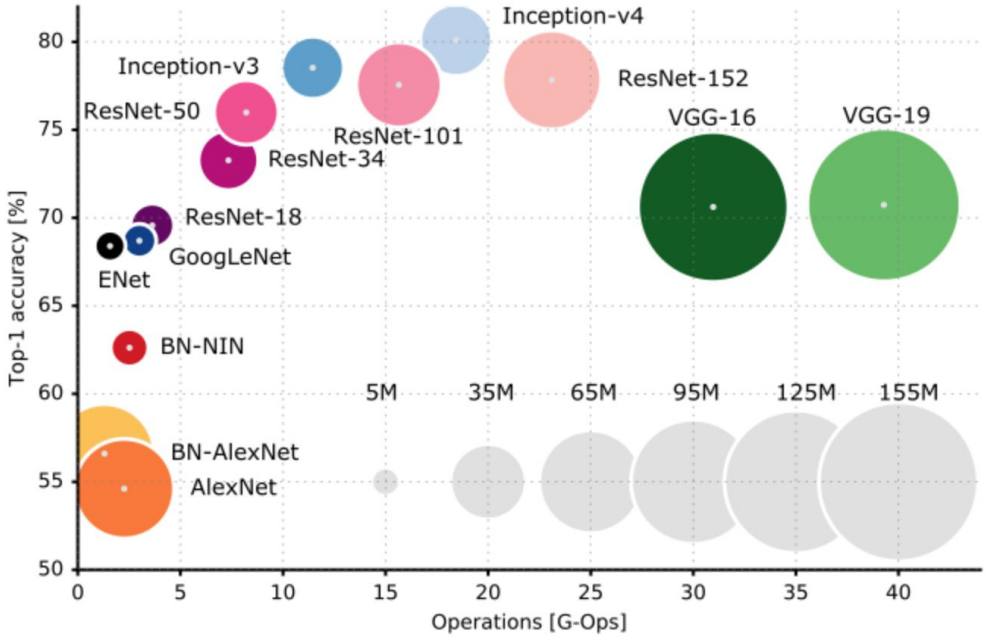


Figure: popular network comparison

Here is a graph showing that some popular network with accuracy and number of G for data flow and architecture size required for memory of computer. Since we don't have much restriction on memory, we prefer to use VGG in our project as fundamental network.

VGG16 (also called OxfordNet) is a convolutional neural network architecture named after the Visual Geometry Group from Oxford, who developed it. It was used to win the ILSVR (ImageNet) competition in 2014. To this day is it still considered to be an excellent vision model, although it has been somewhat outperformed by more recent advances such as Inception and ResNet.

In our project, the input to our ConvNets is a fixed-size 224 × 224 RGB image. The only preprocessing we do is subtracting the mean RGB value, computed on the training set, from each pixel. The image is passed through a stack of convolutional (conv.) layers, where we use filters with a very small receptive field: 3 × 3 (which is the smallest size to capture the notion of left/right, up/down, center). In one of the configurations we also utilize 1 × 1 convolution filters, which can be seen as a linear transformation of the input channels (followed by non-linearity). The convolution stride is fixed to 1 pixel; the spatial padding of conv. layer input is such that the

spatial resolution is preserved after convolution, i.e. the padding is 1 pixel for 3 × 3 conv. layers. Spatial pooling is carried out by five max-pooling layers, which follow some of the conv. layers (not all the conv. layers are followed by max-pooling). Max-pooling is performed over a 2 × 2 pixel window, with stride 2. A stack of convolutional layers (which has a different depth in different architectures) is followed by three Fully-Connected (FC) layers: the first two have 4096 channels each, the third performs 1000- way ILSVRC classification and thus contains 1000 channels (one for each class). The final layer is the soft-max layer. The configuration of the fully connected layers is the same in all networks. All hidden layers are equipped with the rectification (ReLU (Krizhevsky et al., 2012)) non-linearity. We note that none of our networks (except for one) contain Local Response Normalisation (LRN) normalisation (Krizhevsky et al., 2012): as will be shown in Sect. 4, such normalisation does not improve the performance on the ILSVRC dataset, but leads to increased memory consumption and computation time. Where applicable, the parameters for the LRN layer are those of (Krizhevsky et al., 2012).

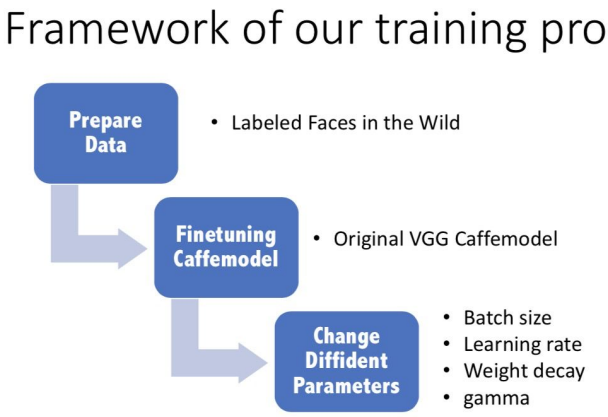


Figure: Framework of Training Process

From the graph above of training process framework, we can obviously check that we use transfer learning during our model training. Finetuning caffemodel weighs heavily in this project and changing with different parameters is also a big portion among all related work.

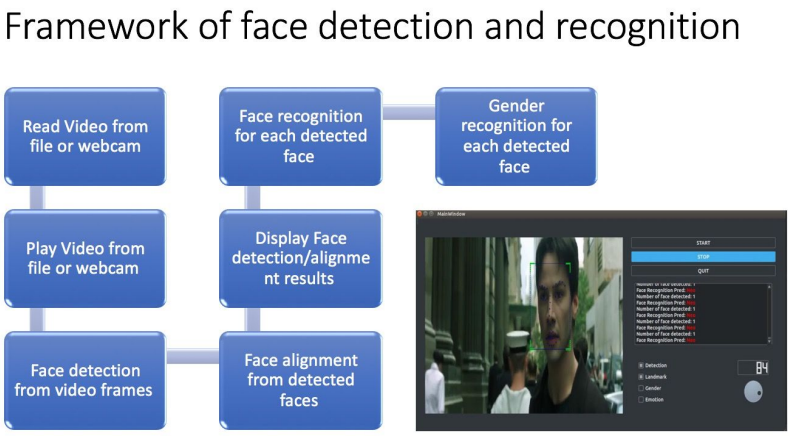


Figure: framework of face detection and recognition

Dlib is a modern C++ toolkit containing machine learning algorithms and tools for creating complex software in C++ to solve real world problems. It is used in both industry and academia in a wide range of domains including robotics, embedded devices, mobile phones, and large high performance computing environments. It has following specialities.

- Open Source
- Complete and precise documentation for every class and function
- Lots of example programs are provided
- High Quality Code
- Machine Learning Algorithms

In our project, we use Dlib to do face detection and show landmarks of facial features like eyebrow, eyes, noses, etc..

Dlib face detector works by using HOG and SVM. HOG face detector was trained by 3000 images from the LFW dataset and training process is only about 3 minutes. It is a real-time detector for image data smaller than 640*480.

Working procedures of Dlib face detector are: load datasets/images, upsample datasets/images, flip datasets/images, train SVM: input images and face box, and test. During these procedures, we use Image Pyramid to handle faces that are too small/large.

Pictures below show an example of HOG (image features)

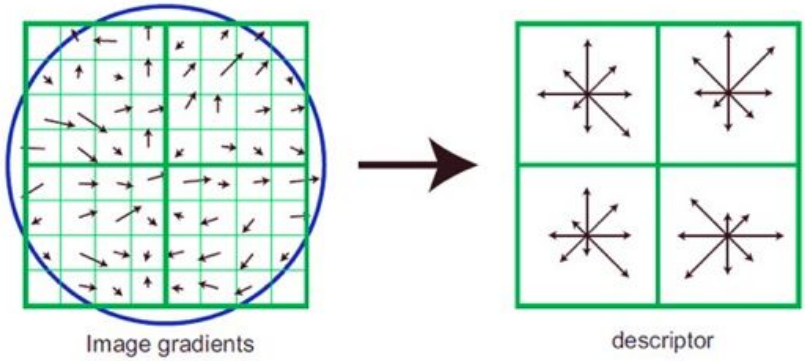


Figure: Generate HOG

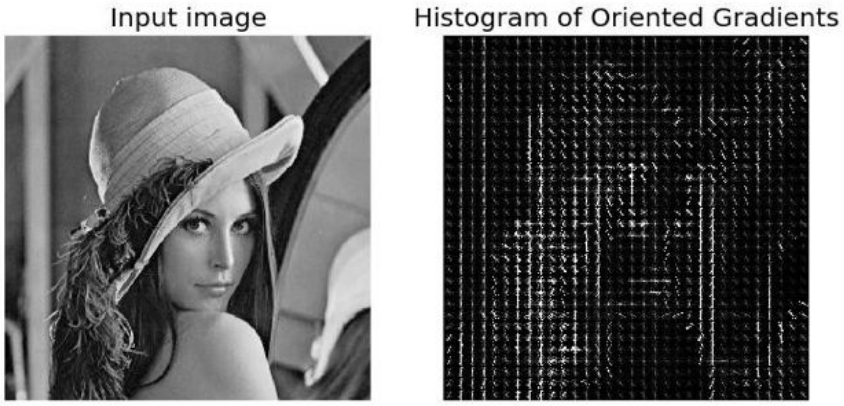


Figure: Input Image and Image Feature

After loading an image into our program, we use image pyramid to shrink or enlarge the image for face detection. We have a 80*80 sliding window which scans all over the image to find a human face if there exists one.

Figure: Example of sliding window

PyQt is a GUI widgets toolkit and it is one of the most powerful and popular cross-platform GUI library. Unlike a console mode application, which is executed in a sequential manner, a GUI based application is event driven. Functions or methods are executed in response to user's actions like clicking on a button, selecting an item from a collection or a mouse click etc., called events.

Experimental setup

We use the pretrained VGG caffemodel to do finetuning.Fine-tuning takes an already learned model, adapts the architecture, and resumes training from the already learned model weights.The LFW images of the Style dataset are visually very similar to the VGG dataset, on which the VGG was trained. Since that model works well for face recognition, we'd like to use this architecture for our style classifier.
Firstly, we choose the people who have more than 20 graphs in LFW dataset. We use python to generate dataset and give the label for each graph. The label start form 0.

```
The number of indentifies: 62
The number of trainset: 2392
The number of testset: 631
Train set and testset have been divided
```

Figure : load data

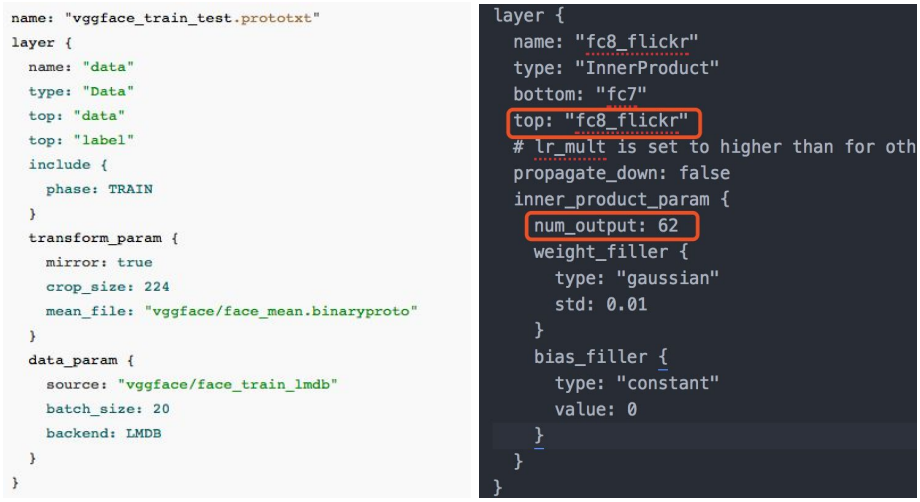
Secondly, we generate LMDB file and Mean file. Because VGG only use the graph which size is 224*224, we resize all the graphs.

```
echo "Creating train lmdb..."

GLOG_logtostderr=1 $TOOLS/convert_imageset.bin \
    --resize_height=$RESIZE_HEIGHT \
    --resize_width=$RESIZE_WIDTH \
    --shuffle \
    $TRAIN_DATA_ROOT \
    $DATA/train.txt \
    $EXAMPLE/face_train_lmdb
```

```
#!/usr/bin/env sh
# Compute the mean image from the imagenet training lmdb
# N.B. this is available in data/ilsvrc12

EXAMPLE=vggface
DATA=vggface
TOOLS=./build/tools

$TOOLS/compute_image_mean $EXAMPLE/face_train_lmdb \
  $DATA/face_mean.binaryproto

echo "Done."
```

Figure : generate LMDB and mean file

Thirdly, we reshape input layer and batch size. After copy the original caffe net, we change the input layer to use our own data and mean file.Because we are predicting 62 classes instead of a 2622, we do need to change the last layer in the model. Therefore, we change the name of the last layer from fc8 to fc8_flickr in our prototxt.



```
name: "vggface_train_test.prototxt"
layer {
  name: "data"
  type: "Data"
  top: "data"
  top: "label"
  include {
    phase: TRAIN
  }
  transform_param {
    mirror: true
    crop_size: 224
    mean_file: "vggface/face_mean.binaryproto"
  }
  data_param {
    source: "vggface/face_train_lmdb"
    batch_size: 20
    backend: LMDB
  }
}
```

```
layer {
  name: "fc8_flickr"
  type: "InnerProduct"
  bottom: "fc7"
  top: "fc8_flickr"
  # lr_mult is set to higher than for oth
  propagate_down: false
  inner_product_param {
    num_output: 62
    weight_filler {
      type: "gaussian"
      std: 0.01
    }
    bias_filler {
      type: "constant"
      value: 0
    }
  }
}
```

Figure : change the original deep learning net

Then, we write our own solver.prototxt. We also decrease the overall learning rate base_lr in the solver prototxt, but boost the lr_multon the newly introduced layer. The idea is to have the rest of the model change very slowly with new data, but let the new layer learn fast. Additionally, we set stepsize in the solver to a lower value than if we were training from scratch, since we're virtually far along in training and therefore want the learning rate to go down faster. Note that we could also entirely prevent fine-tuning of all layers other than fc8_flickr by setting their lr_mult to 0.

```
net: "vggface/vggface_train_test.prototxt"
test_iter: 500
test_interval: 500
test_initialization: false
display: 40
average_loss: 40
base_lr: 0.00005
lr_policy: "step"
stepsize: 320000
gamma: 0.96
max_iter: 1000
momentum: 0.9
weight_decay: 0.0002
snapshot: 500
snapshot_prefix: "vggface/mymodel"
solver_mode: GPU
```

Figure : solver.prototxt

Finally, we write the batch file to finetuning the caffe model, we provide the weights argument to the caffe train command, the pretrained weights will be loaded into our model, matching layers by name.

When we do the recognition process, we extraction features from the 'fc7' layer.
First, we load the database file and forward into the deep learning net, after we get the features of each graph and we store all the label of each graph. Secondly, we load a new graph to the net,

after we got the features, we compute distance to all the labeled features. We think the biggest one correspond the people we recognized.

We used minibatches, we compare the batch size within 16,32,64. The minibatches can't be too small or too large, use the large batchsize will slow the training process and not utilize GPU, however, too large batchsize will cause overfitting. Consider of our dataset and gpu condition, we experience three different batch size.

Because the original VGG Net has a good performance, when we finetune it using our data, we change the value of the learning rate, test_iter and test_interval. We just make them little smaller to get fast training process.

Our method is to hold out a portion of the data, by using a random split(An 70-30 split is generally preferred with 70% of the data used for training). We keep the hold out set until the end and once the model is built we evaluate learned models using this test set. This gives us a general idea of how the model will perform on data on which we actually want to make predictions.

Results

Our results will be delivered as three parts as follows. The first part is the general training process screenshot. We can see details with learning rate specifically to an iteration and training loss at the same time. Also, every 20 iteration we perform a test on test dataset to check on the accuracy and testing loss.

```
I1204 22:09:23.292197  3018 solver.cpp:330] Iteration 1380, Testing net (#0)
I1204 22:09:36.211046  3018 solver.cpp:397]     Test net output #0: accuracy = 0
.973684
I1204 22:09:36.211107  3018 solver.cpp:397]     Test net output #1: loss = 0.194
236 (* 1 = 0.194236 loss)
I1204 22:09:43.634727  3018 solver.cpp:330] Iteration 1400, Testing net (#0)
I1204 22:09:56.564894  3018 solver.cpp:397]     Test net output #0: accuracy = 0
.978618
I1204 22:09:56.565052  3018 solver.cpp:397]     Test net output #1: loss = 0.188
31 (* 1 = 0.18831 loss)
I1204 22:09:56.907537  3018 solver.cpp:218] Iteration 1400 (0.982596 iter/s, 40.
7085s/40 iters), loss = 0.354825
I1204 22:09:56.907598  3018 solver.cpp:237]     Train net output #0: loss = 0.43
255 (* 1 = 0.43255 loss)
I1204 22:09:56.907613  3018 sgd_solver.cpp:105] Iteration 1400, lr = 5e-05
I1204 22:10:03.985793  3018 solver.cpp:330] Iteration 1420, Testing net (#0)
I1204 22:10:13.309342  3030 data_layer.cpp:73] Restarting data prefetching from
start.
I1204 22:10:16.894234  3018 solver.cpp:397]     Test net output #0: accuracy = 0
.972039
I1204 22:10:16.894311  3018 solver.cpp:397]     Test net output #1: loss = 0.186
871 (* 1 = 0.186871 loss)
I1204 22:10:24.329870  3018 solver.cpp:330] Iteration 1440, Testing net (#0)
```

Figure : training process

To make this network more clarified, we printed out feature maps of first convolution layer using a sample image.

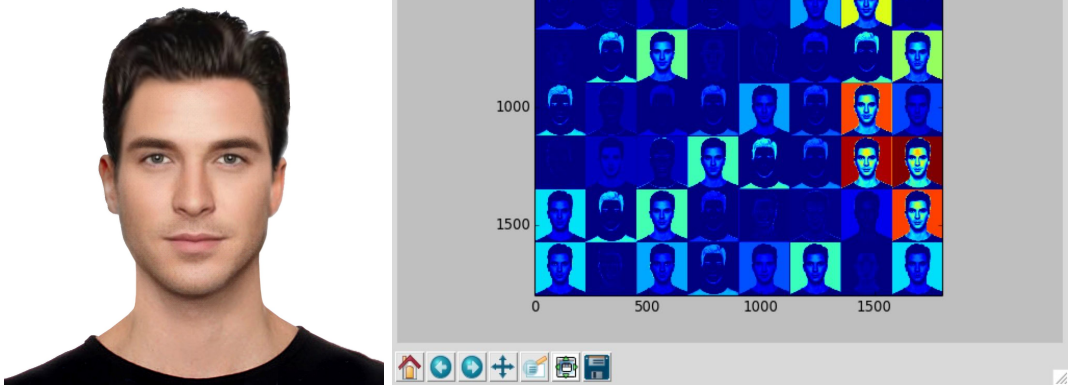


Figure: sample image and its feature maps of conv1 layer

Next part is tuning with different parameters including batch size, learning rate and weight decay. It's well known that network is working with a bunch of parameters not only the three we

mentioned above. But to explain everything as details as we can, we just pick these parameters for tuning. For batch size, we tested on 16, 32, and 64. The test accuracy and training loss graphs are as follows:

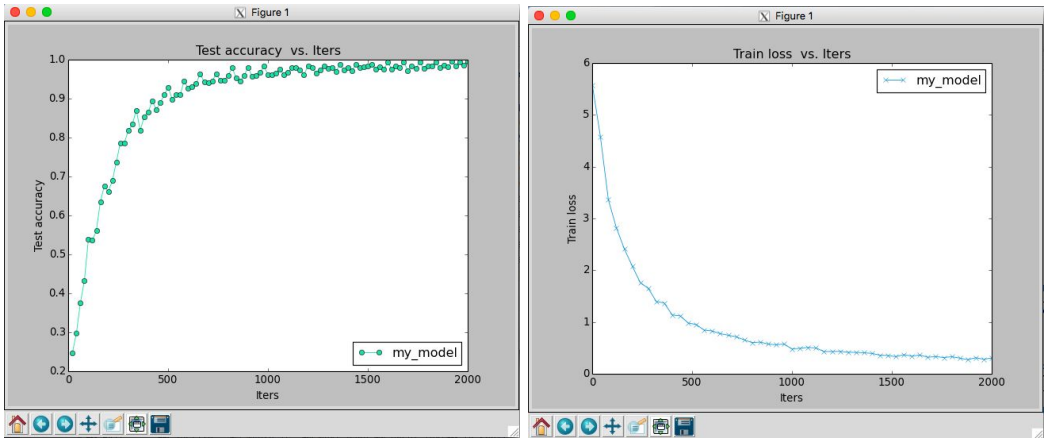


Figure: Test accuracy and train loss for batch size 16

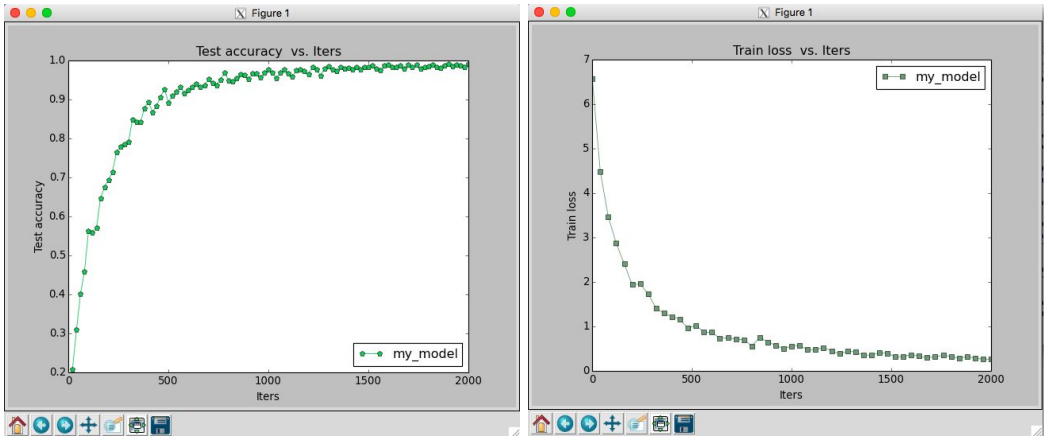


Figure: Test accuracy and train loss for batch size 32

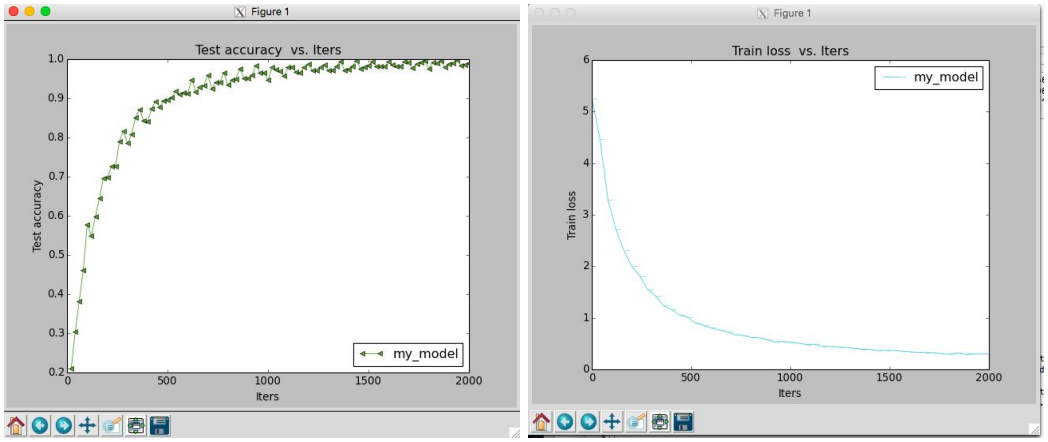


Figure: Test accuracy and train loss for batch size 64

We compared the runtime at the same time showing below. It's necessary to make sure as long as we changed the batch size in the train.prototxt, test_iter in the solver.prototxt should be updated based on the equation of test_iter * batch_size(test) = # of test data.

Batch size	16	32	64
Runtime	35'13''	45'44''	62'45''

Figure: runtime for different batch size

Even we updated the testing iteration, we still can see the significant difference on the runtime but the test accuracy doesn't have much better result.

For experimenting with different learning rates, it is valuable to point out what is a good learning rate respecting to training loss as the figure shows.

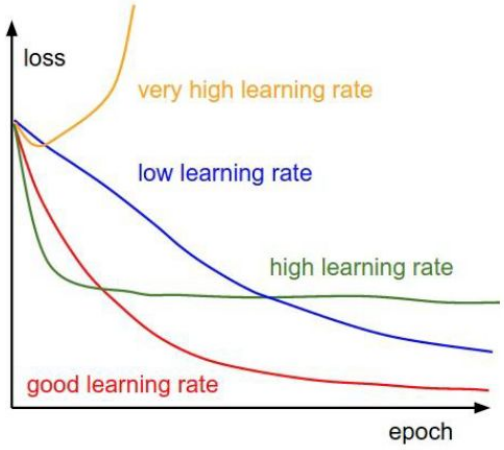


Figure: sample training loss with different learning rate

Here are all graphs of experimenting with different learning rates:

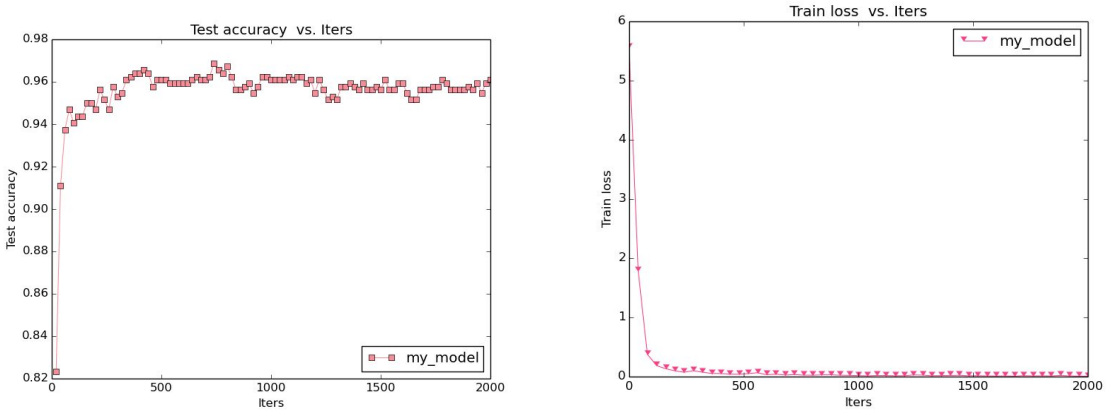


Figure: Test accuracy and train loss for learning rate 0.002

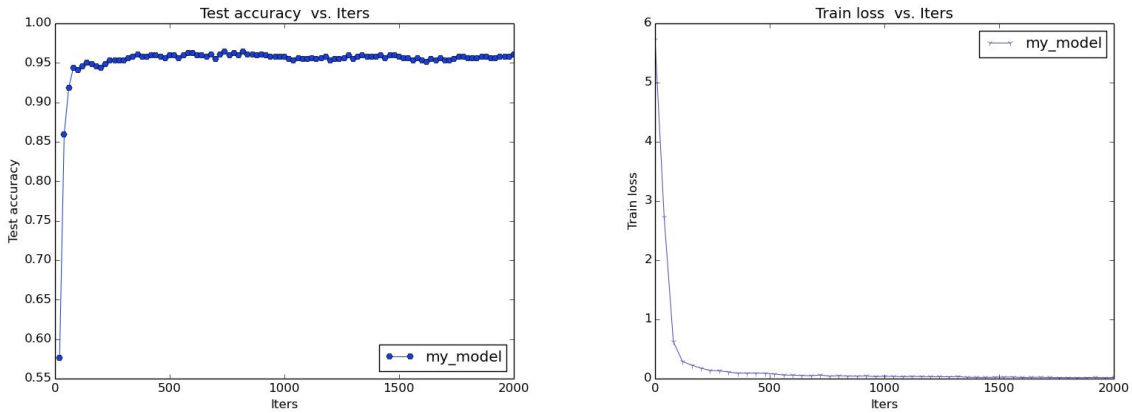


Figure: Test accuracy and train loss for learning rate 0.0005

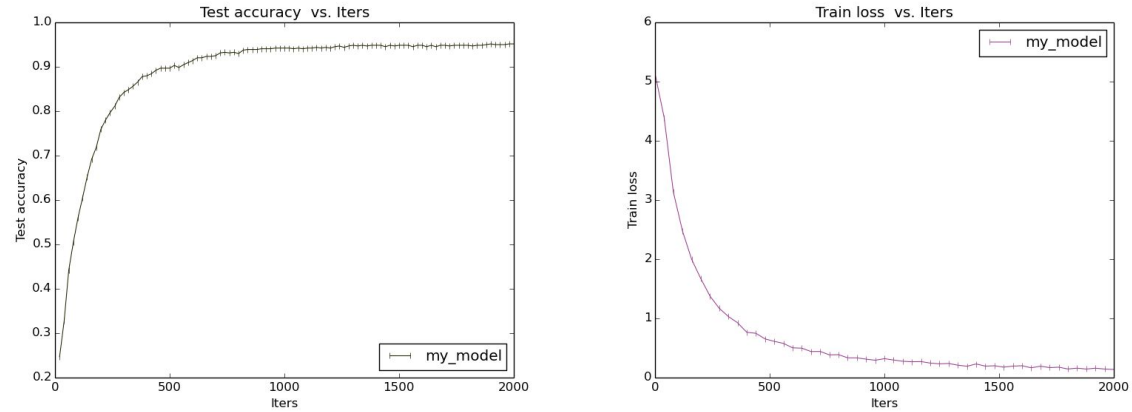


Figure: Test accuracy and train loss for learning rate 0.00005

lr	0.002	0.0005	0.00005
Runtime	43’50’’	44’31’’	44’17’’

Figure: runtime for different learning rate

We don’t see significant difference on the runtime but the test accuracy varies at some level. The test accuracy with learning rate 0.002 does show vibrating even after 1,000 iterations. When test accuracy with 0.0005, we easily can see the test accuracy started to converge at a such early time about iteration 100 comparing to learning rate 0.00005. Although the convergence for learning rate 0.00005 is not significant till 1,200, we still cannot make a conclusion saying that learning rate 0.0005 is better than 0.00005. We do need more investigation on these two comparison. The convergence speed showing slower in the latter one might be due to the learning rate is too small to be converging at an such early time.

Weight decay is not a familiar parameter before doing parameter tuning. It might be useful to do some background introduction on this parameter just in case. Weight decay governs the regularization term and will cause weights to exponentially decay to zero in the weight updating rule. We tested different weight decay but we didn’t see any valuable result that can be delivered here.

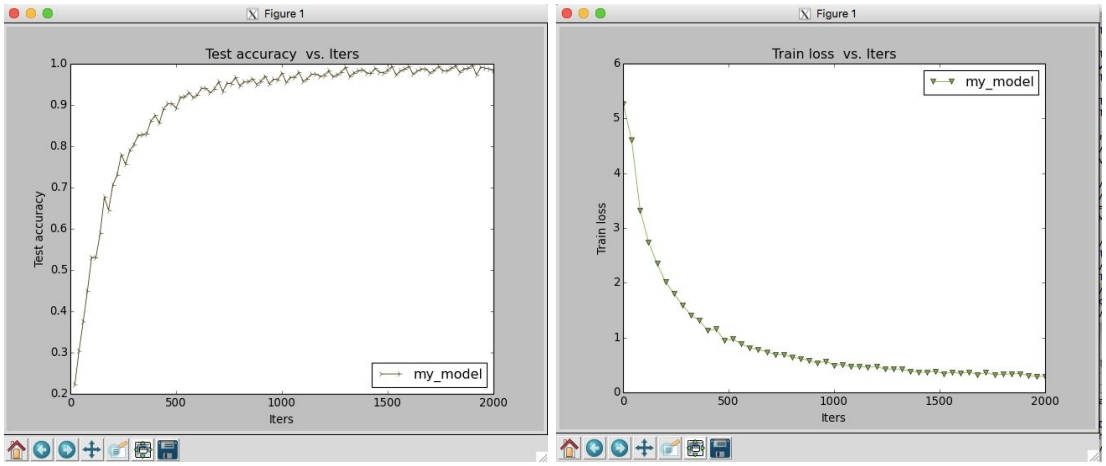


Figure: partial experiment with weight decay

After fine tuning the network and other classification, we could test the whole project accuracy. Firstly we choose 112 people randomly and load their photos in our database,our deep learning net extract the features of these people and store their names. Then we give the model 112 photos of that 112 people the network has never see. We calculate the accuracy that it can recognize these people correctly.

As we can see in the graph below, our model calculate the distance of features between the classify graph and the whole label set. For example, if we have the distance matrix in the below graph, our model will choose the biggest one, and use its index to find the label of this person in the label set.

After testing all the 112 people, we achieved 86.6% test accuracy as following figure shows. We think our network performed well in the face recognition process.

```
dist = [[ 0.21917489  0.05773356  0.16708018  0.13545972  0.17004506  0.05213141
   0.19499691  0.15056314  0.11646565  0.06948718  0.16897358  0.14512467
   0.18320443  0.1962413   0.09443896  0.0389665   0.05758876  0.17135039
   0.12668589  0.16746578  0.17431845  0.10151809  0.0692554   0.2064155
   0.26151061  0.14894994  0.11259218  0.08869986  0.07381545  0.17078491
   0.13999055  0.06270022  0.22352257  0.25827     0.10306538  0.20801196
   0.11331968  0.31712586  0.19625342  0.11887167  0.13941522  0.29654884
   0.16423388  0.23625067  0.14061202  0.11645377  0.23431967  0.03084561
   0.05277288  0.13885383  0.23383656  0.20541006  0.08240889  0.07768483
   0.11506366  0.23963137  0.18182957  0.10726433  0.33374012  0.05918356
   0.15981258  0.15842222  0.09731645  0.23997371  0.06733095  0.18068427
   0.08533545  0.21641159  0.09491615  0.14890461  0.20535678  0.20535892
   0.04088144  0.12123128  0.11595124  0.25758931  0.20340519  0.13152534
   0.04255678  0.25922266  0.05693578  0.22898617  0.10672144  0.13798463
   0.1067066   0.16139294  0.16558957  0.49543777  0.08449372  0.15857381
   0.17398608  0.09088323  0.17024262  0.29006946  0.15878707  0.095877
   0.12782766  0.20642497  0.19294694  0.17900679  0.13886493  0.31870496
   0.21348558  0.14296825  0.09607635  0.22108078  0.09640243  0.38031217
   0.09791578  0.13600276  0.24347593  0.57106006]]
we think these people are:
['Stranger', 'Stranger', 'Ai_Sugiyama', 'Alan_Greenspan', 'Alastair_Campbell', 'Allyson_Felix', 'Angela_Merkel', 'St
ranger', 'Arnoldo_Aleman', 'Ashanti', 'Benazir_Bhutto', 'Stranger', 'Stranger', 'Bertie_Ahern', 'Stranger', 'Carla_D
el_Ponte', 'Carrie-Anne_Moss', 'Catherine_Deneuve', 'Stranger', 'Chen_Shui-bian', 'Choi_Sung-hong', 'Christine_Baumg
artner', 'Ciro_Gomes', 'Clara_Harris', 'Claudia_Pechstein', 'Colin_Montgomerie', 'Cruz_Bustamante', 'David_Anderson'
, 'David_Heymann', 'David_Trimble', 'Denzel_Washington', 'Stranger', 'Elizabeth_Hurley', 'Elizabeth_Smart', 'Emanuel
_Ginobili', 'Emma_Watson', 'Enrique_Bolanos', 'Erika_Harold', 'Frank_Solich', 'Stranger', 'George_Lopez', 'George_Pa
taki', 'Gil_de_Ferran', 'Stranger', 'Gregg_Popovich', 'Habib_Rizieq', 'Hal_Gehman', 'Heidi_Klum', 'Hitomi_Soga', 'Ja
ke_Gyllenhaal', 'James_Wolfensohn', 'Stranger', 'Joseph_Biden', 'Kalpana_Chawla', 'Kevin_Spacey', 'King_Abdullah_II'
, 'Kristanna_Loken', 'Kurt_Warner', 'Lance_Bass', 'LeBron_James', 'Lucy_Liu', 'Ludivine_Sagnier', 'Luis_Gonzalez_Mac
chi', 'Madonna', 'Marcelo_Rios', 'Maria_Soledad_Alvear_Valenzuela', 'Mark_Hurlbert', 'Martha_Stewart', 'Stranger', '
Martina_McBride', 'Michael_Chiklis', 'Michael_Phelps', 'Michael_Powell', 'Michelle_Yeoh', 'Mick_Jagger', 'Mireya_Mos
coso', 'Stranger', 'Nadia_Petrova', 'Natalie_Maines', 'Nia_Vardalos', 'Nick_Nolte', 'Oswaldo_Paya', 'Pamela_Anderson
', 'Pedro_Malan', 'Peter_Struck', 'Prince_Charles', 'Princess_Caroline', 'Queen_Rania', 'Rainer_Schuettler', 'Robert
_Kocharian', 'Robert_Mueller', 'Roger_Moore', 'Russell_Simmons', 'Scott_McClellan', 'Scott_Peterson', 'Stranger', 'S
tranger', 'Stranger', 'Sourav_Ganguly', 'Steffi_Graf', 'Steve_Nash', 'Thabo_Mbeki', 'Tim_Robbins', 'Tom_Crean', 'Tom
_Harkin', 'Vanessa_Redgrave', 'Vicente_Fernandez', 'Victoria_Clarke', 'Wayne_Ferreira', 'William_Macy', 'Woody_Allen
', 'Xanana_Gusmao']

accuracy of this project :
0.866071428571
classify process has been done!
```

Figure : project accuracy

We also have demonstration of our whole project. In this demonstration, we read a video and recognize the face's name and gender in real time. We also can change the threshold in the lower right corner if we want, it depends on how strict we want.

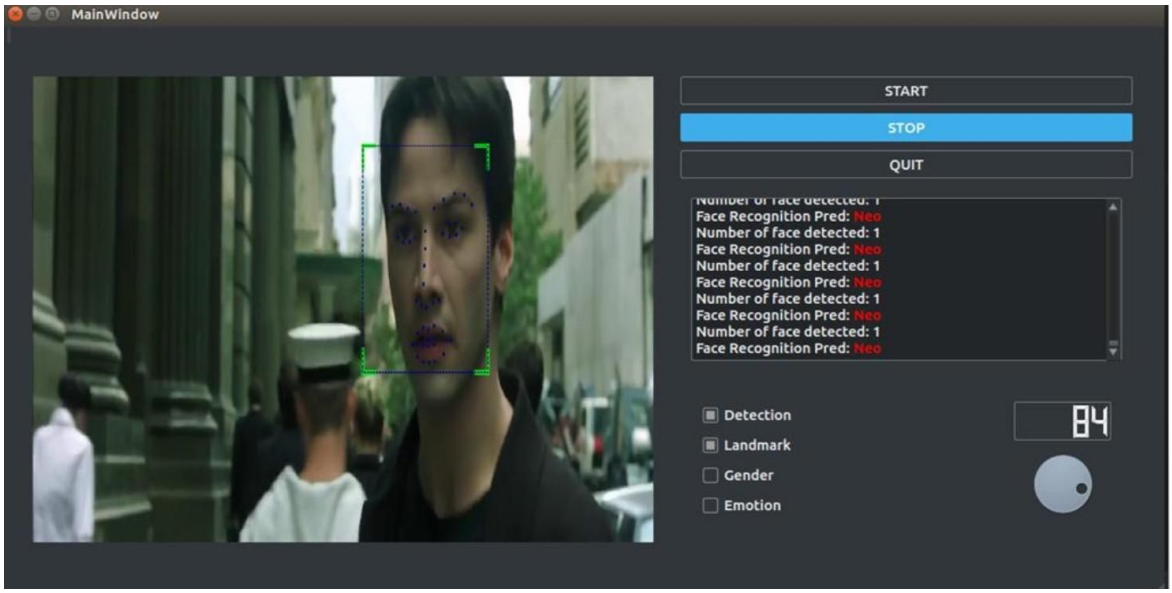


Figure : demonstration

Summary and conclusions

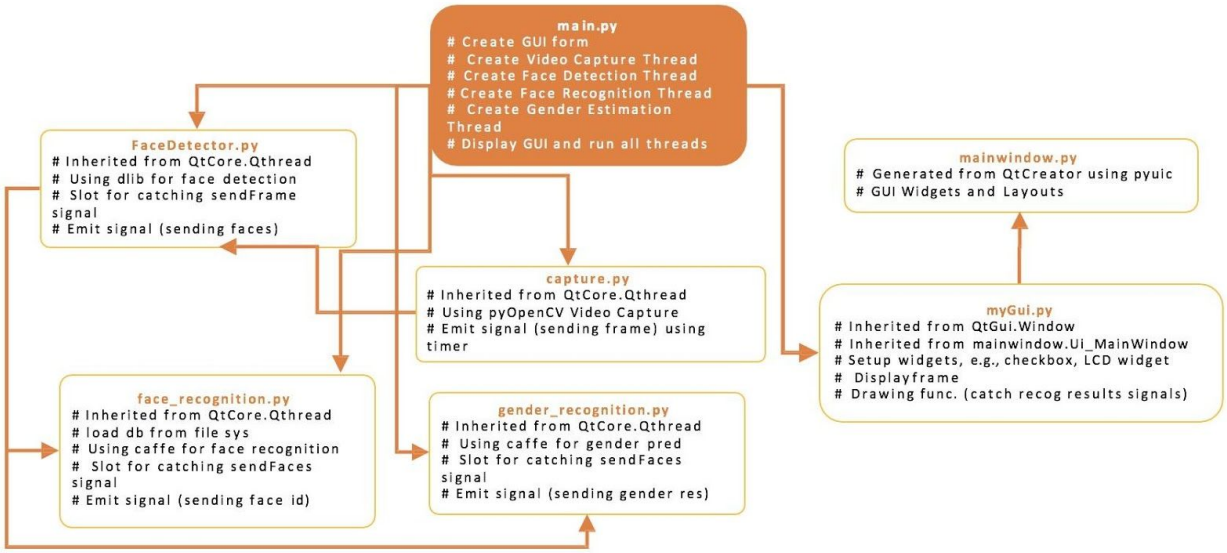


Figure: Frame of Project

Picture above shows module integration of our whole project. Our main python file is main.py, which connects functions like importing video/image, detecting faces, recognizing faces, recognizing gender, and creating GUI together. At the end, it sends all data to myGUI and following with mainwindow.

The results of project is acceptable at this point. We successfully trained a caffemodel with 86.6% accuracy for face recognition, and created a graphic user interface to allow users uploading their own video to do real-time face recognition.

We learned the whole process of doing face detection and face recognition with deep learning networks. The most difficult part was finetuning a caffemodel with a pre-trained caffemodel published by professionals or experienced research teams. Doing right data preparation and fully understanding caffe framework are preconditions of successfully finish our project.

Using similar method as doing face recognition, we can also do emotion recognition, gesture recognition, and age recognition etc. with enough labeled training data.

Also, as accuracy of our fintuned caffemodel is only 86.6%, we can use other dataset to further finetune the model to improve performance of face recognition.

References

Paper and link:

1. https://medium.com/@ageitgey/machine-learning-is-fun-part-4-modern-face-recognition-with-deep-learning-c3cffc121d78
2. https://cmusatyalab.github.io/openface/
3. https://github.com/ydwen/caffe-face
4. http://caffe.berkeleyvision.org/

Data:

5. http://vis-www.cs.umass.edu/lfw/
6. Gary B. Huang, Manu Ramesh, Tamara Berg, and Erik Learned-Miller. Labeled Faces in the Wild: A Database for Studying Face Recognition in Unconstrained Environments. University of Massachusetts, Amherst, Technical Report 07-49, October, 2007.
7. http://blog.dlib.net/
8. Ankan Bansal, Anirudh Nanduri, Carlos D Castillo, Rajeev Ranjan, and Rama Chellappa, UMDFaces: An Annotated Face Dataset for Training Deep Networks, Arxiv preprint, 2016.
9. Ankan Bansal, Carlos Castillo, Rajeev Ranjan, and Rama Chellappa, The Do's and Don'ts for CNN-based Face Verification, Arxiv preprint, 2017.
10. http://x-algo.cn/wp-content/uploads/2017/01/VERY-DEEP-CONVOLUTIONAL-NETWORK-SFOR-LARGE-SCALE-IMAGE-RECOGNITION.pdf

Model:

11. http://www.robots.ox.ac.uk/~vgg/software/vgg_face/
12. http://dlib.net/
13. K. Simonyan, A. Zisserman Very Deep Convolutional Networks for Large-Scale Image Recognition arXiv technical report, 2014
14. https://blog.keras.io/how-convolutional-neural-networks-see-the-world.html
15. http://caffe.berkeleyvision.org/gathered/examples/finetune_flickr_style.html

Visualization:

16. https://www.tutorialspoint.com/pyqt/
17. https://github.com/opencv

A separate appendix should contain documented computer listings.

Please check *VGG full network.pdf*.