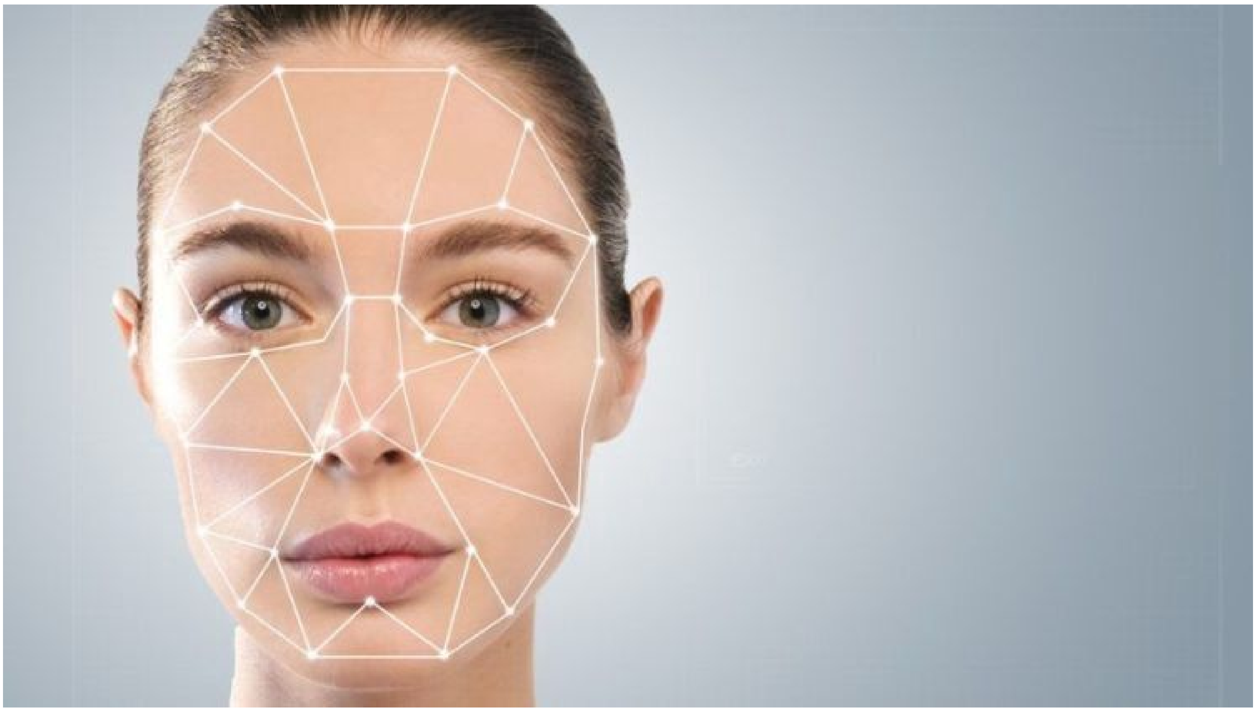
Pattern Recognition Systems, Problem Solving using Pattern Recognition

# Face Recognition

A Classification Problem




Authors: team **NTY**

Ng Siew Pheng (A0198525R)

Tea Lee Seng (A0198538J)

Yang Xiaoyan (A0056720L)

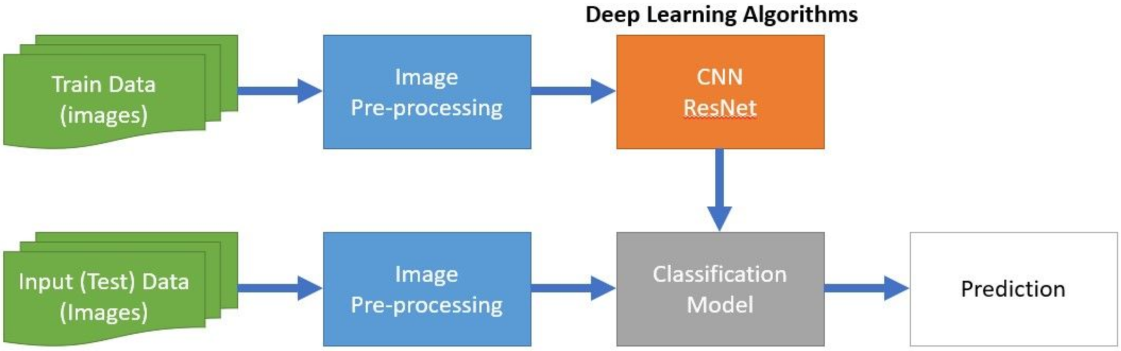
**Table of contents**

# 1. Introduction

A facial recognition system is a technology capable of identifying or verifying a person from a digital image or a video frame from a video source. Facial recognition is a Biometric Artificial Intelligence based application that can uniquely identify a person by analysing patterns based on the person's facial textures and shapes.

Face recognition can be applied Security, Healthcare, Marketing etc areas.

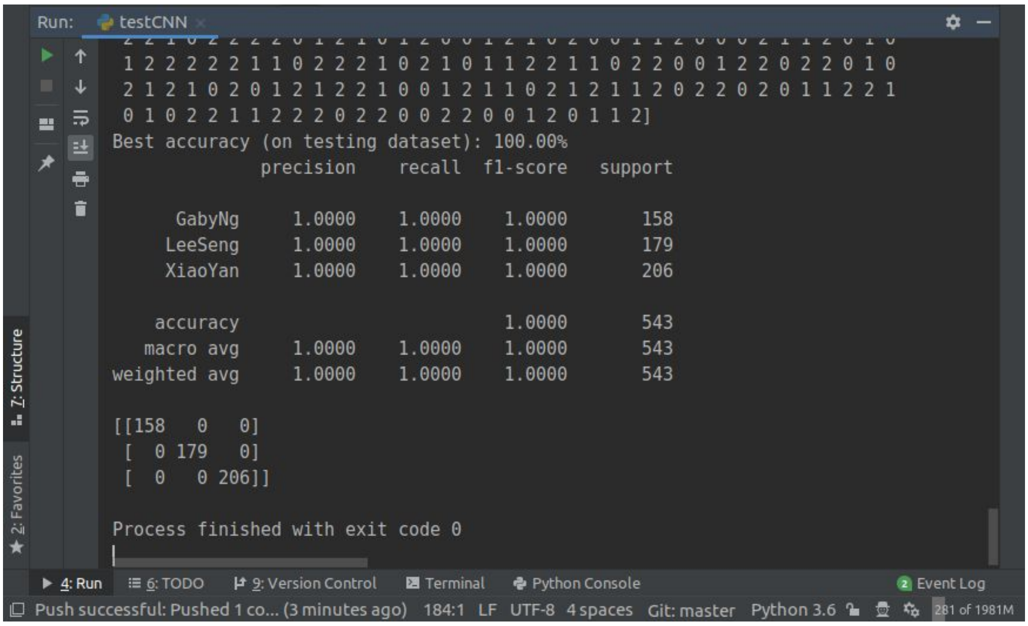
In this project, we will build a facial recognition system by applying deep learning algorithms to identify three people's faces. We will be looking at using CNN and ResNet and determine which model is better.



# 2. Train, Validation and Test Strategy

The data set we are using for this image classification problem, are 2D face images from 3 different people. We record selfie videos of each person from different angles. Each video file is tagged to a particular person. There are about 1800 to 2500  images for each person to perform training. The videos can be downloaded from http://home.leeseng.tech/rawData-20190910.zip

Initially, we apply 70% training, 20% validation and 10% testing data strategy. The result is too good to be true for the 10% testing data.




We suspect the 10% test data is too similar due to the case that we are using data from videos.

A separate set of 21images, from the same 3 people, were used for testing. And these test images are not part of the same video image series. For example, the background might be totally different. This will allow us to test the accuracy of our model when encountering data that it have not been trained with.

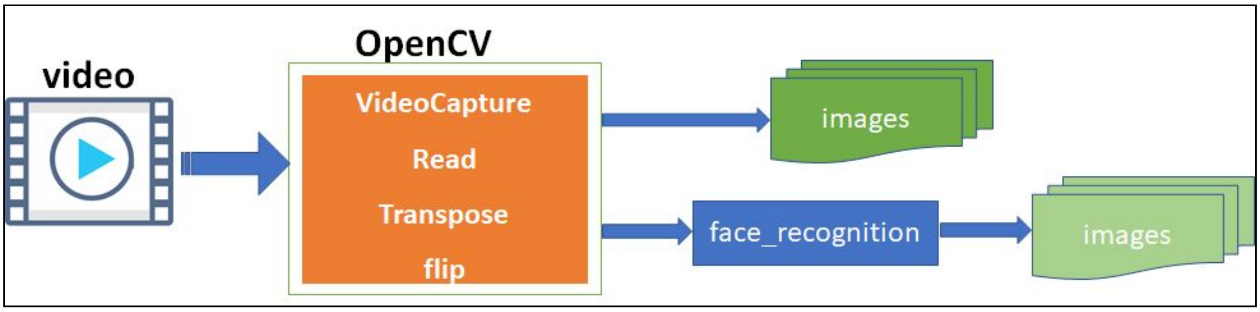
# 3. Data preprocessing

Before we can start training our model, we need to perform data pre-processing to our input data.

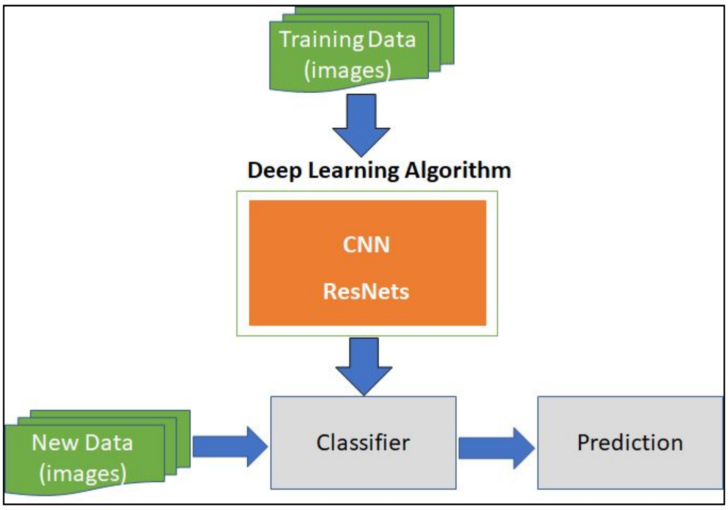
We extracted our images from 2 source formats, namely video (MP4) format and photo (JPEG) format. In order to extract 2D images from the video at a specific frame per second, we created a pre-processing utility, based on OpenCV library. The extracted images are captured in folder, data-full.

In addition to using these 2D images, we also utilized library, [face_recognition](), to crop the face from these 2D images. But there is a small problem. We noticed that if we tried detecting the face using the 2D image in its existing orientation, we weren't able to detect and extract the face. This seems to be due to mobile phone model that we used to capture our own selfie videos. Face clipping data was decided due to less than satisfactory result from normal selfie videos.

Thus, in the face extraction utility, videoToImages.py, if we are unable to detect any faces at 0 degrees, we will rotate the image by 90 degrees and try to detect the face detection. This step is repeated until either a face is detected or the image has completed a full rotation. The extracted face images are captured into folder, data-face. Full training/testing set, with minor cleaning from wrong face cropping, can be downloaded at
http://home.leeseng.tech/training-data-20190925.zip 2.7GB



# 4. Deep learning algorithms




## 4.1. Image Data Generator

Before loading the image data to the deep learning algorithms, we need to perform data augmentation on the training images.

We will perform data augmentation using 3 different Image Data Generators, as shown below, to determine which will help train our model, to yield the highest and consistent accuracy.

| Parameter | Rescale Generator | Samplewise Generator | Samplewise + Rescale Generator |
|---|---|---|---|
| samplewise_center | | True | True |
| samplewise_std_normalization | | True | True |
| rescale | 1.0/255 | | 1.0/255 |
| width_shift_range | 0.1 | 0.1 | 0.1 |
| height_shift_range | 0.1 | 0.1 | 0.1 |
| rotation_range | 20 | 20 | 20 |
| zoom_range | 0.1 | 0.1 | 0.1 |
| shear_range | 0.15 | 0.15 | 0.15 |
| horizontal_flip | True | True | True |
| vertical_flip | False | False | False |
| fill_mode | nearest | nearest | nearest |

## 4.2. Convolutional Neural Network (CNN)

We have constructed the following CNN model and using an input image of pixel 128 x 128.

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 126, 126, 32)      896
_________________________________________________________________
activation (Activation)      (None, 126, 126, 32)      0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 124, 124, 64)      18496
_________________________________________________________________
activation_1 (Activation)    (None, 124, 124, 64)      0
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 62, 62, 64)        0
_________________________________________________________________
dropout (Dropout)            (None, 62, 62, 64)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 60, 60, 128)       73856
_________________________________________________________________
activation_2 (Activation)    (None, 60, 60, 128)       0
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 30, 30, 128)       0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 28, 28, 256)       295168
_________________________________________________________________
activation_3 (Activation)    (None, 28, 28, 256)       0
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 14, 14, 256)       0
_________________________________________________________________
dropout_1 (Dropout)          (None, 14, 14, 256)       0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 12, 12, 256)       590080
_________________________________________________________________
activation_4 (Activation)    (None, 12, 12, 256)       0
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 6, 6, 256)         0
_________________________________________________________________
dropout_2 (Dropout)          (None, 6, 6, 256)         0
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 4, 4, 512)         1180160
_________________________________________________________________
activation_5 (Activation)    (None, 4, 4, 512)         0
_________________________________________________________________
flatten (Flatten)            (None, 8192)              0
_________________________________________________________________
dense (Dense)                (None, 512)               4194816
_________________________________________________________________
activation_6 (Activation)    (None, 512)               0
_________________________________________________________________
dropout_3 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_1 (Dense)              (None, 3)                 1539
=================================================================
Total params: 6,355,011
Trainable params: 6,355,011
Non-trainable params: 0
```

With this model, we will start training it using the 3 different image data generators, as mentioned in section 5.1.

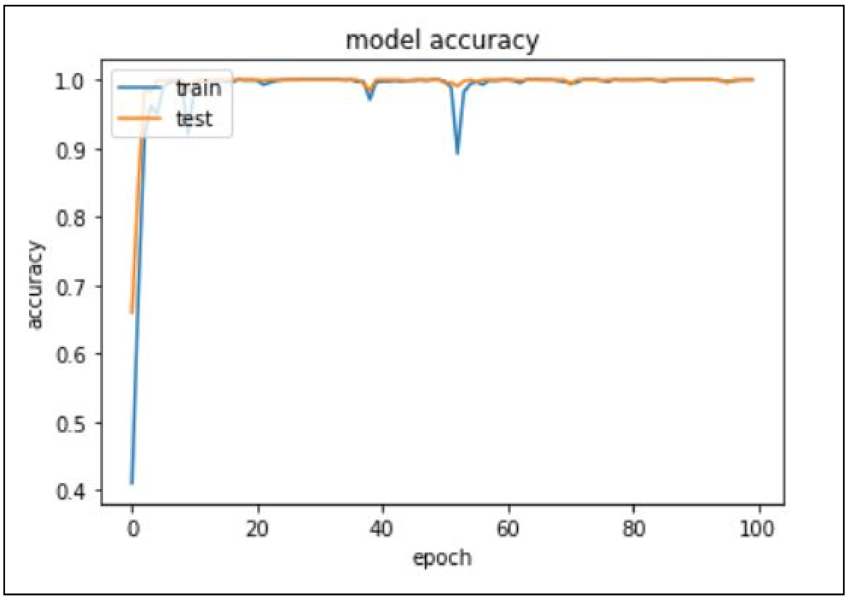
### 4.2.1. Training with full images

- Rescale ImageDataGenerator

```
testout: [0 0 0 0 0 2 2 2 2 2 2 2 2 2 1 1 1 1 1 1 1]
predout: [1 0 1 0 0 1 2 2 1 1 2 2 1 0 1 1 2 2 1 2 0]
Best accuracy (on testing dataset): 47.62%
              precision    recall  f1-score   support

      GabyNg     0.6000    0.6000    0.6000         5
     LeeSeng     0.3333    0.4286    0.3750         7
     XiaoYan     0.5714    0.4444    0.5000         9

    accuracy                         0.4762        21
   macro avg     0.5016    0.4910    0.4917        21
weighted avg     0.4989    0.4762    0.4821        21

[[3 2 0]
 [1 3 3]
 [1 4 4]]
```

### 4.2.2. Training with face only images

- Rescale ImageDataGenerator
  Training Model : [CNN_FaceTrain_Rescale.ipynb](CNN_FaceTrain_Rescale.ipynb)



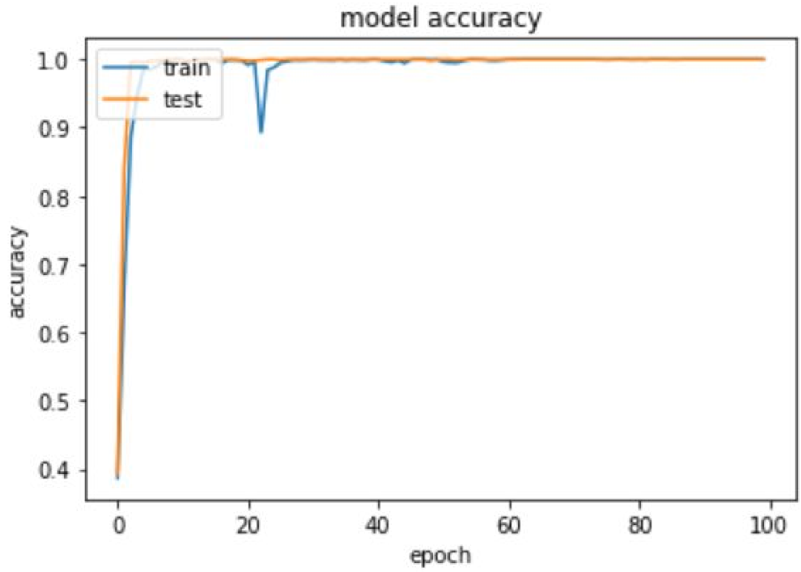
```
test result for CNN using face only images in rescale data generator
Batch size is: 128
testout: [0 0 0 0 0 2 2 2 2 2 2 2 2 2 1 1 1 1 1 1 1]
predout: [1 0 0 0 0 2 2 2 2 2 2 2 2 0 1 0 1 1 1 2 1]
Best accuracy (on test data set): 80.95%
              precision    recall  f1-score   support

      GabyNg     0.6667    0.8000    0.7273         5
     LeeSeng     0.8333    0.7143    0.7692         7
     XiaoYan     0.8889    0.8889    0.8889         9

    accuracy                         0.8095        21
   macro avg     0.7963    0.8011    0.7951        21
weighted avg     0.8175    0.8095    0.8105        21

[[4 1 0]
 [1 5 1]
 [1 0 8]]
```

Based on the result, with batch size set as 128, we wonder what will the impact be we use a lower batch size. The result below shows that accuracy has dropped.

```
testout: [0 0 0 0 0 2 2 2 2 2 2 2 2 2 1 1 1 1 1 1 1]
predout: [0 0 0 2 1 2 0 0 2 1 2 2 0 0 1 0 1 1 1 2 1]
Batch size is 64
Best accuracy (on testing dataset): 57.14%
              precision    recall  f1-score   support

      GabyNg     0.3750    0.6000    0.4615         5
     LeeSeng     0.7143    0.7143    0.7143         7
     XiaoYan     0.6667    0.4444    0.5333         9

    accuracy                         0.5714        21
   macro avg     0.5853    0.5862    0.5697        21
weighted avg     0.6131    0.5714    0.5766        21

[[3 1 1]
 [1 5 1]
 [4 1 4]]
```

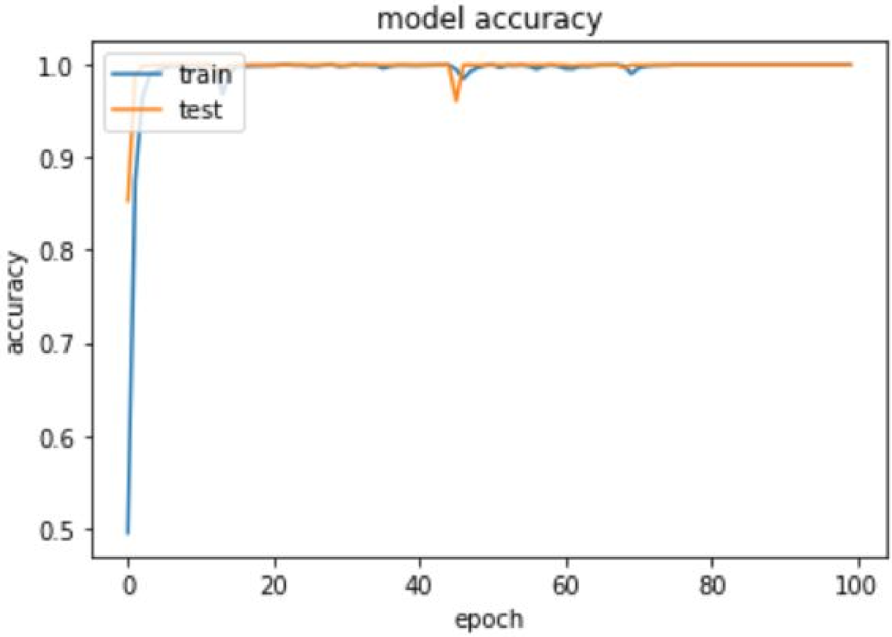
With this result, we have decided to set batch_size for our image data generator to 128.

- Samplewise ImageDataGenerator
  Training Model : [CNN_FaceTrain_Sample.ipynb]()




```
Test Results for CNN using face only images in samplewise data generator
testout: [0 0 0 0 0 2 2 2 2 2 2 2 2 2 1 1 1 1 1 1 1]
predout: [0 0 0 0 0 2 2 0 2 2 2 2 2 0 1 1 1 1 1 0 1]
Batch size is 128
Best accuracy (on testing dataset): 85.71%
              precision    recall  f1-score   support

      GabyNg     0.6250    1.0000    0.7692         5
     LeeSeng     1.0000    0.8571    0.9231         7
     XiaoYan     1.0000    0.7778    0.8750         9

    accuracy                         0.8571        21
   macro avg     0.8750    0.8783    0.8558        21
weighted avg     0.9107    0.8571    0.8658        21

[[5 0 0]
 [1 6 0]
 [2 0 7]]
```

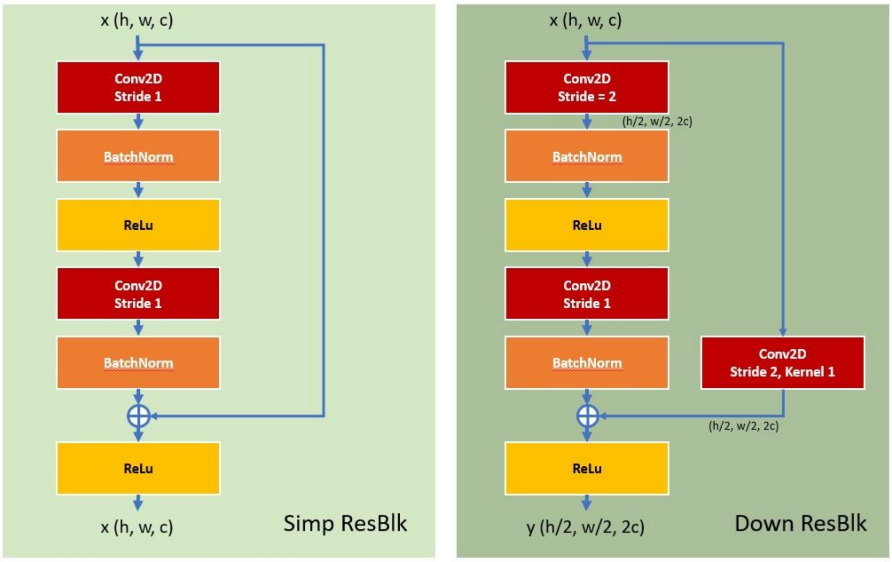
- Samplewise and Rescale ImageDataGenerator
  Training Model : [CNN_FaceTrain_Sample_Rescale.ipynb]

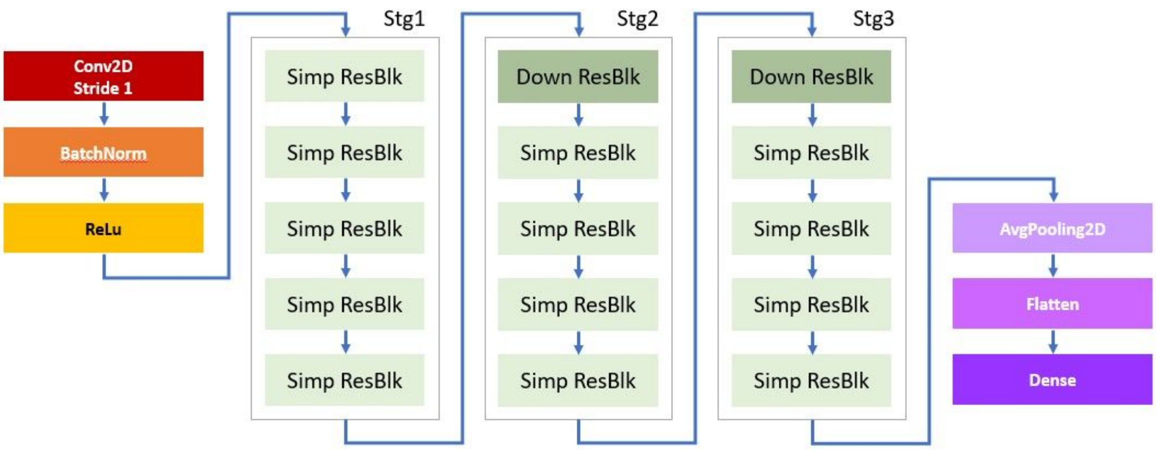



```
Test Results for CNN using face only images in samplewise
and rescale data generator
testout: [0 0 0 0 0 2 2 2 2 2 2 2 2 2 1 1 1 1 1 1 1]
predout: [1 0 0 0 0 2 2 2 2 2 2 2 2 0 1 1 0 1 1 0 1]
Batch size is 128
Best accuracy (on testing dataset): 80.95%
              precision    recall  f1-score   support

      GabyNg     0.5714    0.8000    0.6667         5
     LeeSeng     0.8333    0.7143    0.7692         7
     XiaoYan     1.0000    0.8889    0.9412         9

    accuracy                         0.8095        21
   macro avg     0.8016    0.8011    0.7924        21
weighted avg     0.8424    0.8095    0.8185        21

[[4 1 0]
 [2 5 0]
 [1 0 8]]
```

## 4.3. Residual neural network (ResNet)

We have constructed the following ResNet model and using an input image of pixel 128 x 128.



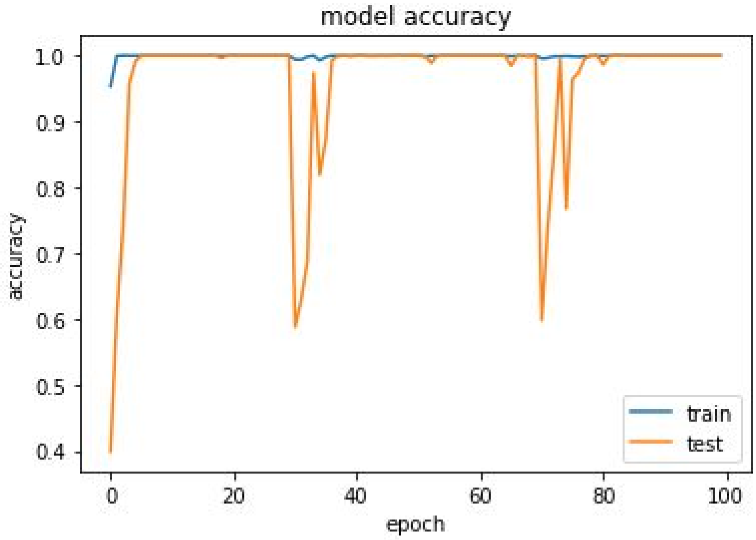

Total params: 473,411
Trainable params: 471,139
Non-trainable params: 2,272

### 4.3.1. Training with full images

- Rescale ImageDataGenerator
  Training Model: [ResNet_FullTrain_Rescale.ipynb]()

```
testout: [0 0 0 0 0 2 2 2 2 2 2 2 2 2 1 1 1 1 1 1 1]
predout: [1 0 1 1 1 2 2 2 2 1 1 1 0 2 1 1 1 1 1 0 1]
Best accuracy (on testing dataset): 57.14%
              precision    recall  f1-score   support

      GabyNg     0.3333    0.2000    0.2500         5
     LeeSeng     0.4615    0.8571    0.6000         7
     XiaoYan     1.0000    0.5556    0.7143         9

    accuracy                         0.5714        21
   macro avg     0.5983    0.5376    0.5214        21
weighted avg     0.6618    0.5714    0.5656        21

[[1 4 0]
 [1 6 0]
 [1 3 5]]
```

- Samplewise ImageDataGenerator
  Training Model: [ResNet_FullTrain_Sample.ipynb]()



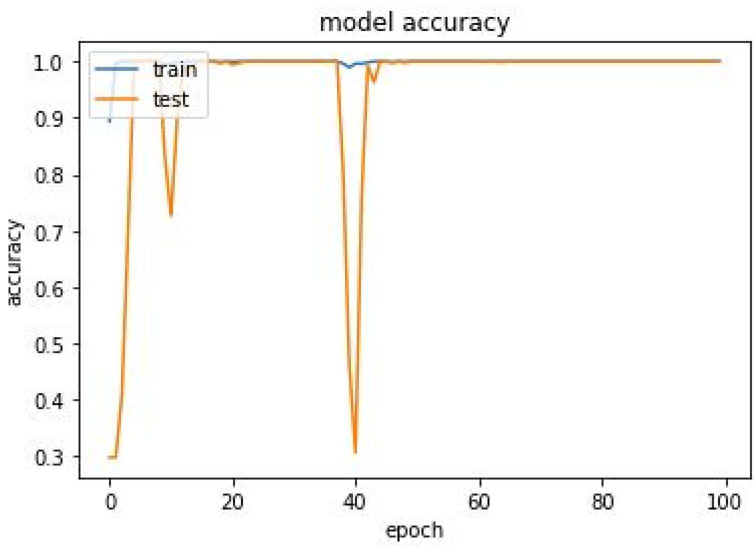

```
testout: [0 0 0 0 0 2 2 2 2 2 2 2 2 2 1 1 1 1 1 1 1]
predout: [0 0 1 1 1 2 1 2 1 1 1 1 0 0 1 0 1 0 1 2 1]
Best accuracy (on testing dataset): 38.10%
              precision    recall  f1-score   support

      GabyNg     0.3333    0.4000    0.3636         5
     LeeSeng     0.3333    0.5714    0.4211         7
     XiaoYan     0.6667    0.2222    0.3333         9

    accuracy                         0.3810        21
   macro avg     0.4444    0.3979    0.3727        21
weighted avg     0.4762    0.3810    0.3698        21

[[2 3 0]
 [2 4 1]
 [2 5 2]]
```

- Samplewise and Rescale ImageDataGenerator

```
testout: [0 0 0 0 0 1 2 2 2 2 2 2 2 2 2]
predout: [1 0 0 1 0 1 2 0 2 2 1 1 1 0 0]
Best accuracy (on testing dataset): 46.67%
              precision    recall  f1-score   support

      GabyNg     0.5000    0.6000    0.5455         5
     LeeSeng     0.1667    1.0000    0.2857         1
     XiaoYan     1.0000    0.3333    0.5000         9

    accuracy                         0.4667        15
   macro avg     0.5556    0.6444    0.4437        15
weighted avg     0.7778    0.4667    0.5009        15

[[3 2 0]
 [0 1 0]
 [3 3 3]]
```

### 4.3.2. Training with face only images

- Rescale ImageDataGenerator
  Training Model: [ResNet_FaceTrain_Rescale.ipynb]()



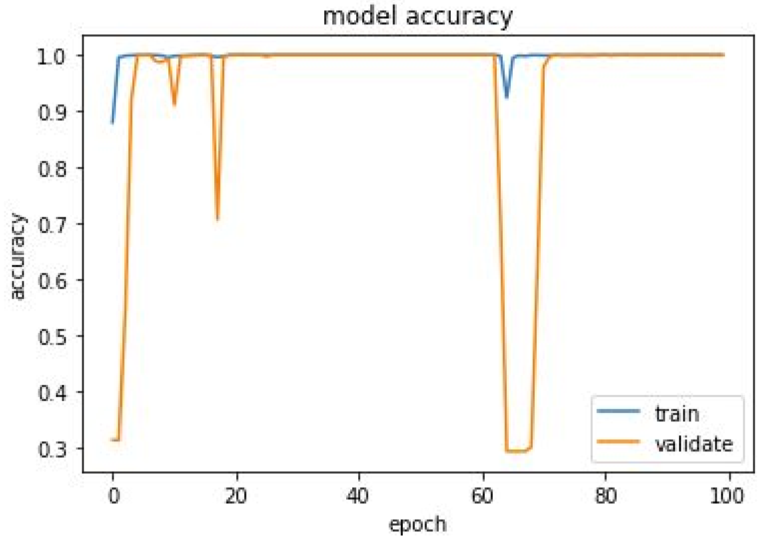

Using the input test images in its original form to perform testing.

```
testout: [0 0 0 0 0 2 2 2 2 2 2 2 2 2 1 1 1 1 1 1 1]
predout: [1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 1 1 1 1]
Batch size is 128
Best accuracy (on testing dataset): 23.81%
              precision    recall  f1-score   support

      GabyNg     0.0000    0.0000    0.0000         5
     LeeSeng     0.2632    0.7143    0.3846         7
     XiaoYan     0.0000    0.0000    0.0000         9

    accuracy                         0.2381        21
   macro avg     0.0877    0.2381    0.1282        21
weighted avg     0.0877    0.2381    0.1282        21

[[0 5 0]
 [2 5 0]
 [0 9 0]]
```

Tested using resultant images, generated after running face extraction utility on the input test images

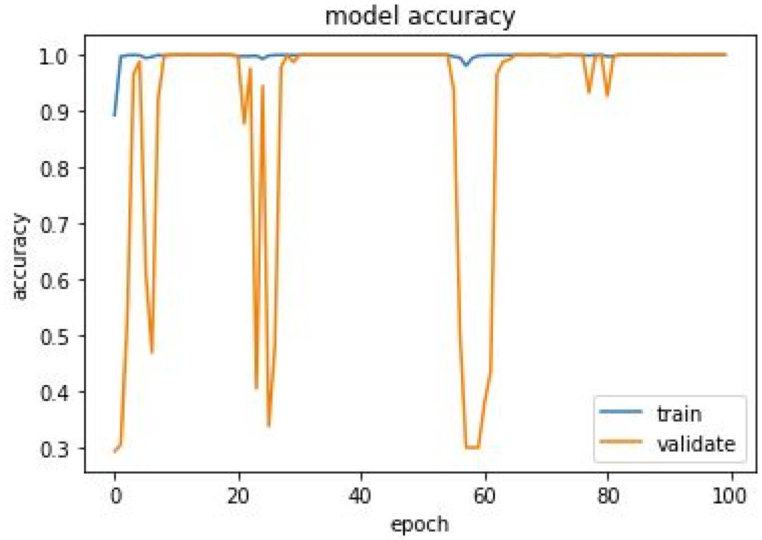
```
testout: [0 0 0 0 0 2 2 2 2 2 2 2 2 2 1 1 1 1 1 1 1]
predout: [1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1]
Batch size is 128
Best accuracy (on testing dataset): 33.33%
              precision    recall  f1-score   support

      GabyNg     0.0000    0.0000    0.0000         5
     LeeSeng     0.3333    1.0000    0.5000         7
     XiaoYan     0.0000    0.0000    0.0000         9

    accuracy                         0.3333        21
   macro avg     0.1111    0.3333    0.1667        21
weighted avg     0.1111    0.3333    0.1667        21

[[0 5 0]
 [0 7 0]
 [0 9 0]]
```

- Samplewise ImageDataGenerator
  Training Model: [ResNet_FaceTrain_Sample.ipynb](ResNet_FaceTrain_Sample.ipynb)




```
Test Results for ResNet using face only images in samplewise data generator
testout: [0 0 0 0 0 2 2 2 2 2 2 2 2 2 1 1 1 1 1 1 1]
predout: [1 0 0 0 0 2 2 0 2 2 0 0 2 0 1 1 0 1 1 0 1]
Batch size is 128
Best accuracy (on testing dataset): 66.67%
              precision    recall  f1-score   support

      GabyNg     0.4000    0.8000    0.5333         5
     LeeSeng     0.8333    0.7143    0.7692         7
     XiaoYan     1.0000    0.5556    0.7143         9

    accuracy                         0.6667        21
   macro avg     0.7444    0.6899    0.6723        21
weighted avg     0.8016    0.6667    0.6895        21

[[4 1 0]
 [2 5 0]
 [4 0 5]]
```

- Samplewise and Rescale ImageDataGenerator
  Training Model: [ResNet_FaceTrain_Sample_and_Rescale.ipynb](ResNet_FaceTrain_Sample_and_Rescale.ipynb)




```
testout: [0 0 0 0 0 2 2 2 2 2 2 2 2 2 1 1 1 1 1 1 1]
predout: [2 0 0 0 0 2 0 0 1 0 0 0 2 0 1 2 1 2 1 0 1]
Batch size is 128
Best accuracy (on testing dataset): 47.62%
              precision    recall  f1-score   support

      GabyNg     0.3636    0.8000    0.5000         5
     LeeSeng     0.8000    0.5714    0.6667         7
     XiaoYan     0.4000    0.2222    0.2857         9

    accuracy                         0.4762        21
   macro avg     0.5212    0.5312    0.4841        21
weighted avg     0.5247    0.4762    0.4637        21

[[4 0 1]
 [1 4 2]
 [6 1 2]]
```

## 4.4. Model Test Result Comparison

Using a set of 21 full 2D images as test set, we performed prediction using the models we have trained so far. Following are the test results we obtained.

| Model | | Train Data | Test Data | |
|---|---|---|---|---|
| | | | Full Image | Face Only |
| CNN | rescale | Full Image | 47.6% | |
| | | Face Only | 47.62% | 80.95% |
| | Samplewise | Full Image | | |
| | | Face Only | 28.57% | 85.7%, 69.57% |
| | Samplewise+ rescale | Full Image | | |
| | | Face Only | 33.33% | 76.19% |
| ResNet | rescale | Full Image | 57% | |
| | | Face Only | 23.81% | 33.3% |
| | Samplewise | Full Image | 67% | |
| | | Face Only | 42.86% | 66.67% |
| | Samplewise+ rescale | Full Image | 47% | |
| | | Face Only | 61.90% | 47.62% |

Observation:

- Using CNN Model and data augmentation on the images to detect and crop out the face image and applying samplewise setting in the data generator, yielded the best accuracy of 85.7%.
- Whereas using full image for both train and test yielded the worst result. Interesting the worst prediction is coming from classification of child pictures in our data sets.
- For the resNet, we are getting better test results with sample wise setting (samplewise_center=True, samplewise_std_normalization=True,) as compared with using rescale (rescale = 1. / 255) the images.

# 5. Challenges

## 5.1. Videos to images

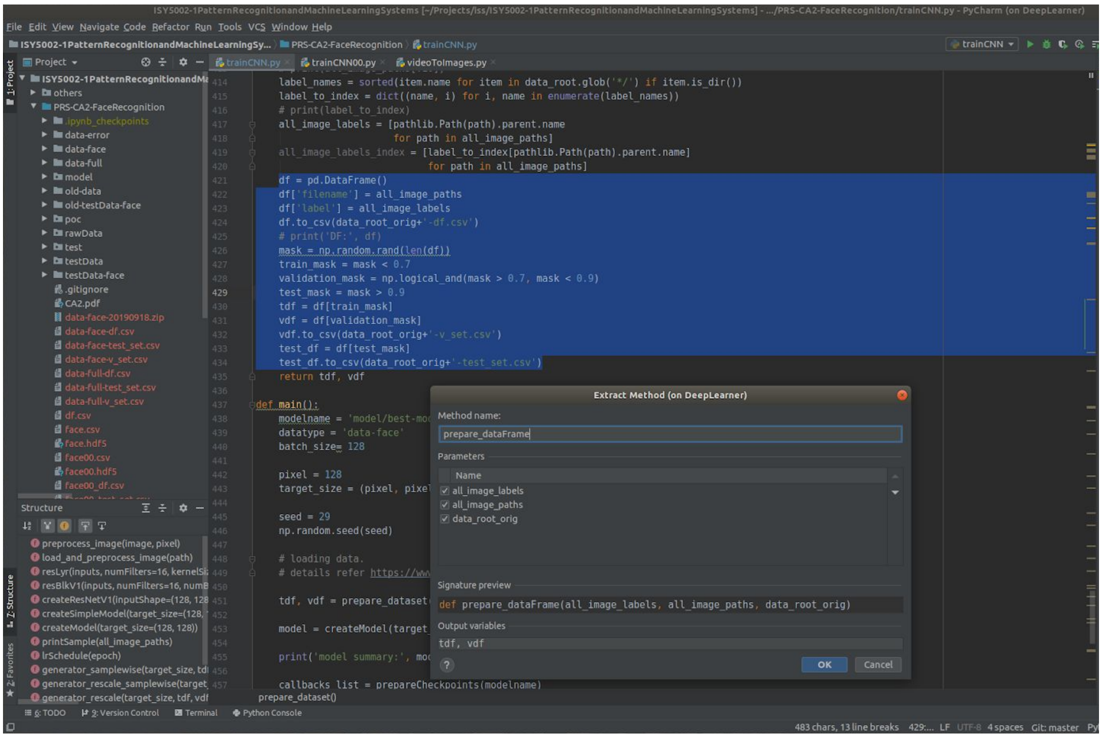
We used mobile phones to capture videos for our face images. Due to different camera app implementation, we have to perform following transformation to get upright photos from video clips.

```
if className in ('XiaoYan', 'GabyNg'):
    image = cv2.transpose(image)
elif className in ('LeeSeng'):
    image = cv2.transpose(image)
    image = cv2.flip(image, 0)
```

## 5.2. The usage of jupyter notebook and code quality

Data project is still a bunch of source code. It is normal to write spaghetti code to try things out and show the result in jupyter notebook. However, it is critical for new developer to avoid using jupyter notebook at all cost, as it doesn't provide refactoring facility. Jupyter notebook is meant for generating research reports. And yet after trying parameters and network setup, all reported figures disappeared into 1 jupyter notebook, and replication of the result requires meddling with parameters again.

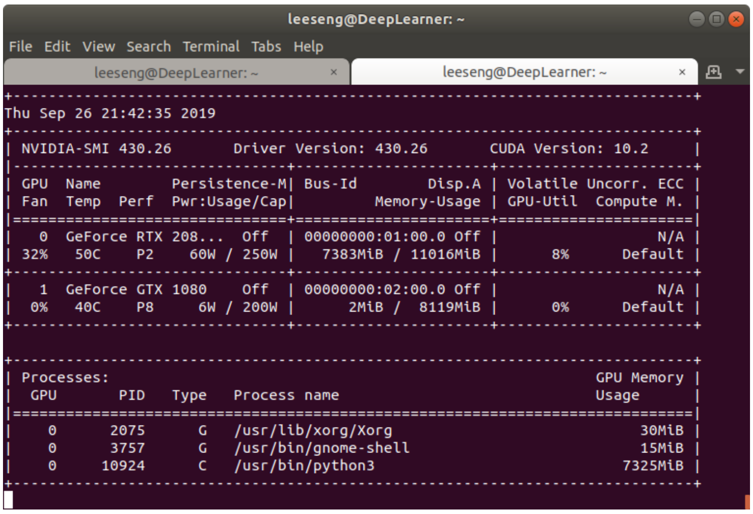
It would be better to use IDE with refactoring facility, e.g. pycharm community edition, to refactor spaghetti code into methods for reusability.



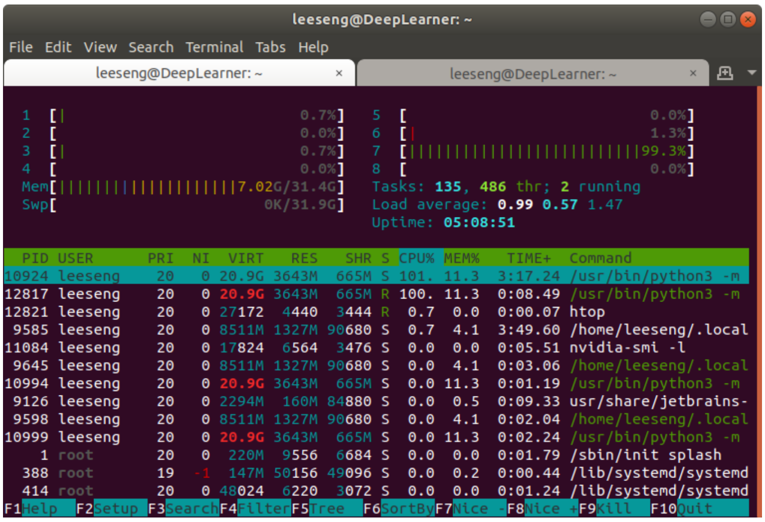

With simple methods, it will be easier to generate reports under jupyter notebook, and produce less errors in various reports under different scenario.

## 5.3. Slower training due to single process image augmentation

We always wonder why low GPU usage during slow training.


```
leeseng@DeepLearner: ~
File  Edit  View  Search  Terminal  Tabs  Help
leeseng@DeepLearner: ~          leeseng@DeepLearner: ~
+-----------------------------------------------------------------------------+
Thu Sep 26 21:42:35 2019
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 430.26       Driver Version: 430.26       CUDA Version: 10.2     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|===============================+======================+======================|
|   0  GeForce RTX 208...  Off  | 00000000:01:00.0 Off |                  N/A |
| 32%   50C    P2    60W / 250W |   7383MiB / 11016MiB |      8%      Default |
+-------------------------------+----------------------+----------------------+
|   1  GeForce GTX 1080    Off  | 00000000:02:00.0 Off |                  N/A |
|  0%   40C    P8     6W / 200W |      2MiB /  8119MiB |      0%      Default |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                       GPU Memory |
|  GPU       PID   Type   Process name                             Usage      |
|=============================================================================|
|    0      2075      G   /usr/lib/xorg/Xorg                            30MiB |
|    0      3757      G   /usr/bin/gnome-shell                          15MiB |
|    0     10924      C   /usr/bin/python3                            7325MiB |
+-----------------------------------------------------------------------------+
```

```
_________________________________________________________________
activation_27 (Activation)   (None, 512)               0
_________________________________________________________________
dropout_16 (Dropout)         (None, 512)               0
_________________________________________________________________
dense_7 (Dense)              (None, 3)                 1539
=================================================================
Total params: 6,355,011
Trainable params: 6,355,011
Non-trainable params: 0
_________________________________________________________________
model summary: None
Found 4488 validated image filenames belonging to 3 classes.
Found 1306 validated image filenames belonging to 3 classes.
Learning rate: 0.001
Epoch 1/50
35/35 [==============================] - 59s 2s/step - loss: 0.9921 - acc: 0.5498 - val_loss: 0.4096 - val_acc:
0.8508
Learning rate: 0.001
Epoch 2/50
35/35 [==============================] - 58s 2s/step - loss: 0.2145 - acc: 0.9220 - val_loss: 0.0585 - val_acc:
0.9875
Learning rate: 0.001
Epoch 3/50
11/35 [========>.....................] - ETA: 27s - loss: 0.0405 - acc: 0.9865
```


```
leeseng@DeepLearner: ~
File  Edit  View  Search  Terminal  Tabs  Help
leeseng@DeepLearner: ~          leeseng@DeepLearner: ~

  1  [|                        0.7%]   5  [                          0.0%]
  2  [                         0.0%]   6  [|                         1.3%]
  3  [|                        0.7%]   7  [||||||||||||||||||||||||||99.3%]
  4  [                         0.0%]   8  [                          0.0%]
  Mem[|||||||||||||||||||||7.02G/31.4G]   Tasks: 135, 486 thr; 2 running
  Swp[                     0K/31.9G]   Load average: 0.99 0.57 1.47
                                       Uptime: 05:08:51

  PID USER      PRI  NI  VIRT   RES   SHR S CPU% MEM%   TIME+  Command
10924 leeseng    20   0 20.9G 3643M  665M S 101. 11.3  3:17.24 /usr/bin/python3 -m
12817 leeseng    20   0 20.9G 3643M  665M R 100. 11.3  0:08.49 /usr/bin/python3 -m
12821 leeseng    20   0 27172  4440  3444 R  0.7  0.0  0:00.07 htop
 9585 leeseng    20   0 8511M 1327M 90680 S  0.7  4.1  3:49.60 /home/leeseng/.local
11084 leeseng    20   0 17824  6564  3476 S  0.0  0.0  0:05.51 nvidia-smi -l
 9645 leeseng    20   0 8511M 1327M 90680 S  0.0  4.1  0:03.06 /home/leeseng/.local
10994 leeseng    20   0 20.9G 3643M  665M S  0.0 11.3  0:01.19 /usr/bin/python3 -m
 9126 leeseng    20   0 2294M  160M 84880 S  0.0  0.5  0:09.33 usr/share/jetbrains-
 9598 leeseng    20   0 8511M 1327M 90680 S  0.0  4.1  0:02.04 /home/leeseng/.local
10999 leeseng    20   0 20.9G 3643M  665M S  0.0 11.3  0:02.24 /usr/bin/python3 -m
    1 root       20   0  220M  9556  6684 S  0.0  0.0  0:01.79 /sbin/init splash
  388 root       19  -1  147M 50156 49096 S  0.0  0.2  0:00.44 /lib/systemd/systemd
  414 root       20   0 48024  6220  3072 S  0.0  0.0  0:01.24 /lib/systemd/systemd
F1Help  F2Setup F3SearchF4FilterF5Tree  F6SortByF7Nice -F8Nice +F9Kill  F10Quit
```

After observe CPU usage, we suspect ImageDataGenerator is single process by default.

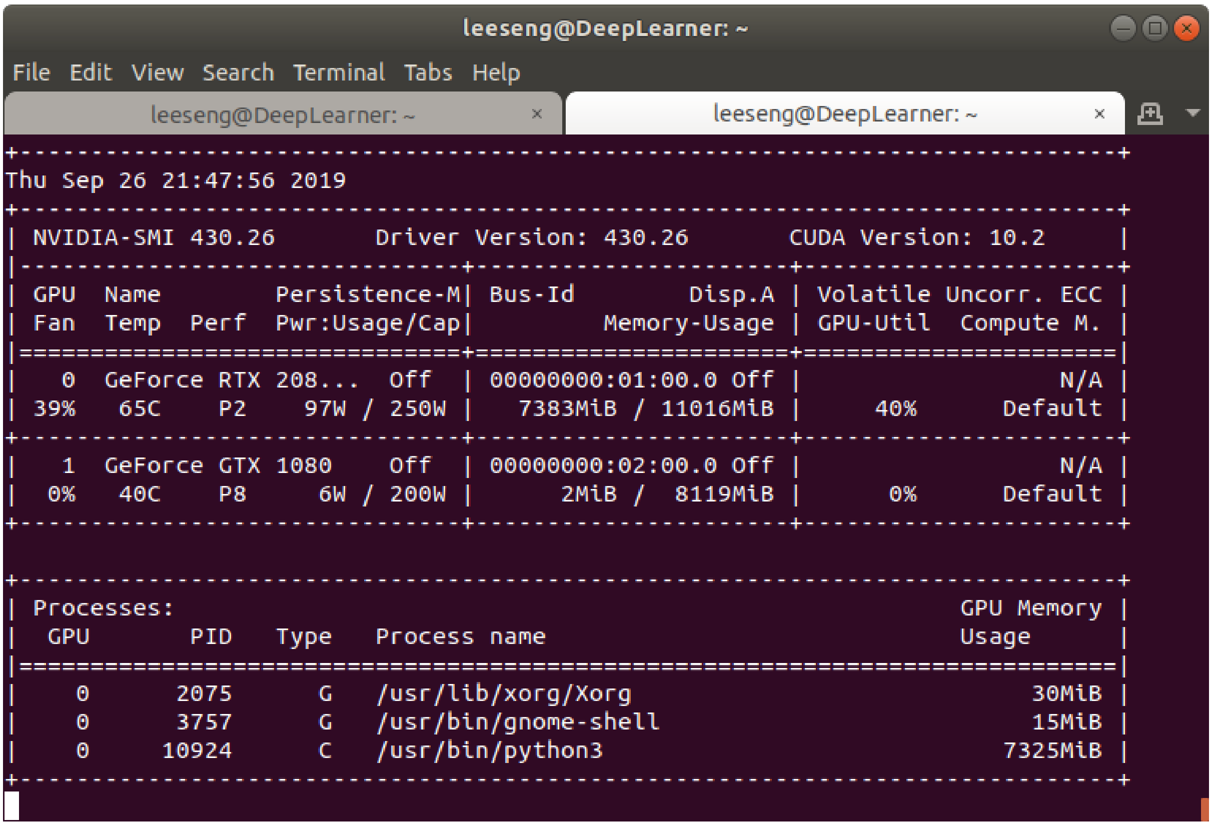
After searching around tensorflow API, we noticed that Model.fit_generator() seems to be able to do multiprocessing.

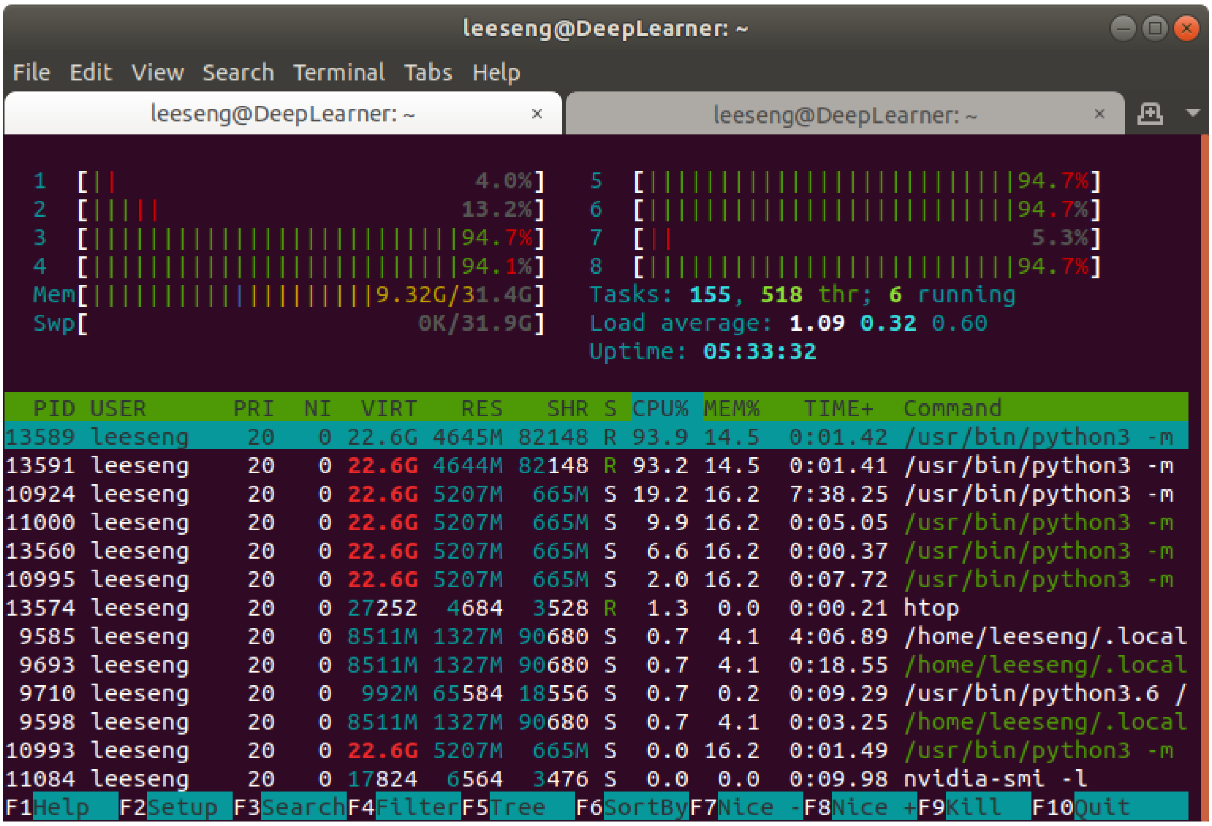
- `workers`: Integer. Maximum number of processes to spin up when using process-based threading. If unspecified, `workers` will default to 1. If 0, will execute the generator on the main thread.
- `use_multiprocessing`: Boolean. If `True`, use process-based threading. If unspecified, `use_multiprocessing` will default to `False`. Note that because this implementation relies on multiprocessing, you should not pass non-picklable arguments to the generator as they can't be passed easily to children processes.

After setting "workers=5, use_multiprocessing=True", we reduce the training time to 1/3 of existing training time, from 58seconds to 18 seconds. Suggests workers number to be "total cpu count - 2". The 2 cpus are reserved for OS and jupyter notebook main process.

```
_________________________________________________________________
dropout_20 (Dropout)         (None, 512)               0
_________________________________________________________________
dense_9 (Dense)              (None, 3)                 1539
=================================================================
Total params: 6,355,011
Trainable params: 6,355,011
Non-trainable params: 0
_________________________________________________________________
model summary: None
Found 4524 validated image filenames belonging to 3 classes.
Found 1297 validated image filenames belonging to 3 classes.
Learning rate: 0.001
Epoch 1/50
35/35 [==============================] - 18s 510ms/step - loss: 0.8433 - acc: 0.6015 - val_loss: 0.2785 - val_a
cc: 0.8852
Learning rate: 0.001
Epoch 2/50
35/35 [==============================] - 18s 517ms/step - loss: 0.1446 - acc: 0.9570 - val_loss: 0.0770 - val_a
cc: 0.9859
Learning rate: 0.001
Epoch 3/50
35/35 [==============================] - 18s 518ms/step - loss: 0.0377 - acc: 0.9879 - val_loss: 0.0203 - val_a
cc: 0.9969
Learning rate: 0.001
Epoch 4/50
 8/35 [=====>........................] - ETA: 11s - loss: 0.0261 - acc: 0.9951
```

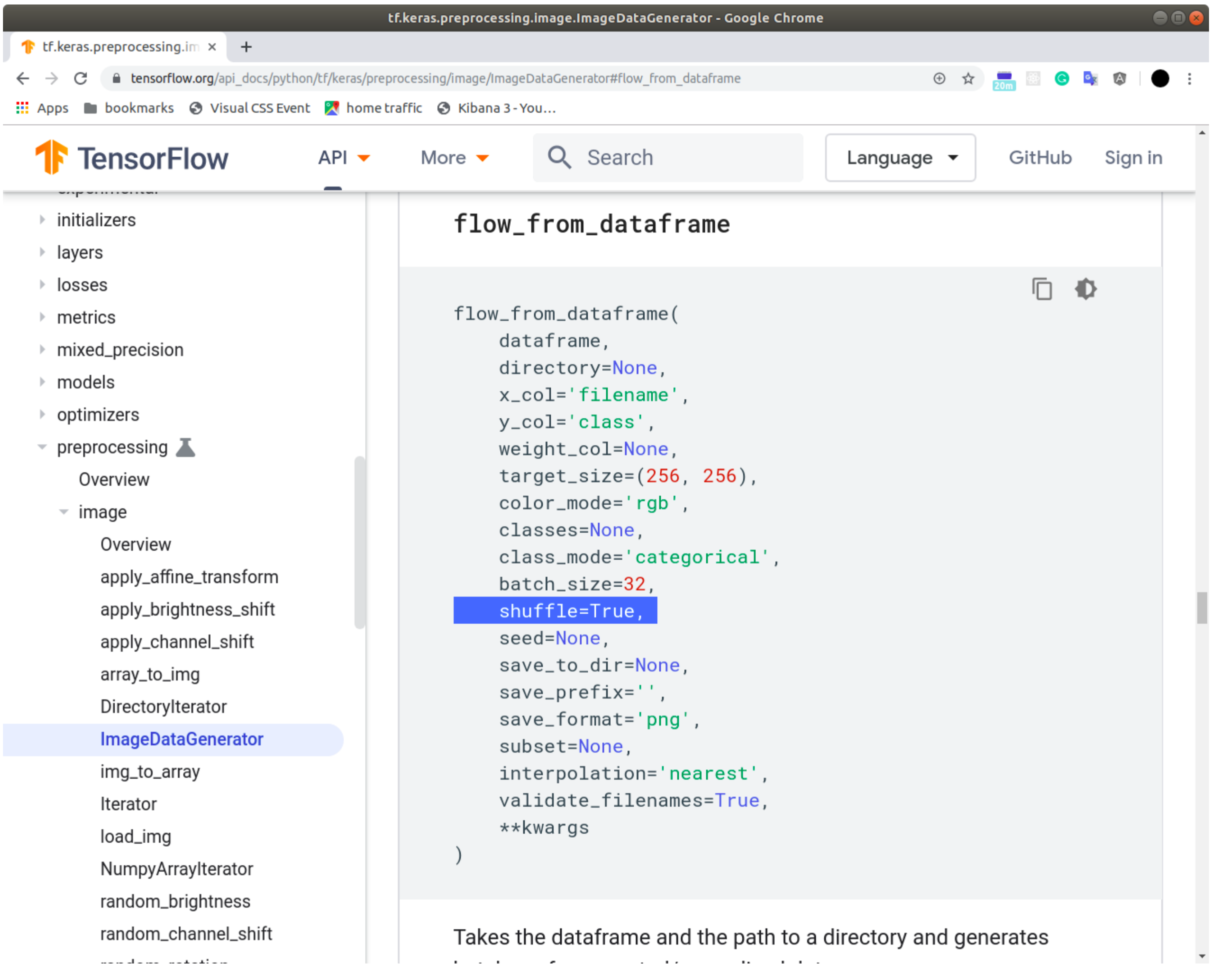
CPU and GPU usage also become more fully utilized.



### 5.4. ImageDataGenerator.flow_from_dataframe(), implicit shuffle=True

This is one of the tricky bugs to catch under testing. The true labels are provided by dataframe, but flow_from_dataframe() API has “shuffle = True” by default. We can only observe random accuracy even with same model for a few rounds of testing. We discover that by applying the validation set images to testing. With validation accuracy of 90% during training, it is weird that we got random accuracy testing score again. After recalling that “shuffle = True” in ImageDataGenerator.flow_from_dataframe(), setting “shuffle=False” resolve this testing bug.



### 5.5. High validation accuracy BUT Low testing accuracy

During our initial training both our model using full images for both training and validation, the validation accuracy is 100%. But when we apply the model for test images prediction, the accuracy is very low, most of the models get accuracy less than 60%. The likely possible reason is due to larger and similar background of faces during selfie video. Therefore, we learn the hard way only during testing phase. Fortunately, we have [face_recognition](https://github.com/ageitgey/face_recognition) API to crop faces for training and testing, which helps to improve our final result accuracy to 85%, from 67%.

## 6. Conclusion

Using a CNN model, we started our training and testing using the 2D images we extracted from our videos. After training the model with different parameters and we could not improve the accuracy above 50%, we realised the 2D images extracted from our video, contains too much background “noise” to the face images. As such, we determined that in order to get a better accuracy of our face recognition, it is important to extract only the face images to train our model.

We got the worst test accuracy for full images especially wrong prediction for the child’s images. The child’s full images’ background maybe is the noise for the training which cause the prediction not accurate problem. That prompts us to consider cropping only face for training. We researched on the Internet and found https://github.com/ageitgey/face_recognition API about face clipping. To remove the noise of the images, we use face crop images for our models training, the test accuracy much better and achieve 85%.

## References:

https://medium.com/@ageitgey/machine-learning-is-fun-part-4-modern-face-recognition-with-deep-learning-c3cffc121d78#.ds8i8oic9