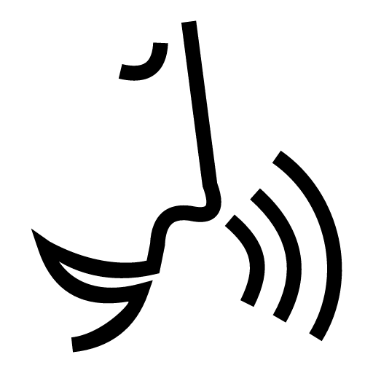
Team 12

Who is Speaking? 

20171151 “Sanghyub Lee” SIT

20172058 “Dongwoo Koo” SIT

20172074 “Khawar Naheem” SIT

20172114 “Sanghun Jeon” SIT

20172125 “Ahmed El Sharkawy” SIT

**Table of Contents**

1. Introduction

1.1 Deep Learning background

1.1.1 Image classification

1.1.1.1 CNN

1.1.1.2 Inception V3

1.1.2 Voice recognition

2. Proposed System

2.1 Environment setup

2.2 Overall structure

2.2.1 Voice signal interface and conversion

2.2.1.1 Voice signal interface

2.2.1.2 Voice signal conversion

2.2.2 Database (DB)

2.2.3 Main algorithm

3. System Evaluation

3.1 Scenarios

3.1.1 Comparison between FFT- and spectrogram-based DB

3.1.2 Accuracy enhancement using bigger DB

3.1.3 Testing accuracy enhancement under restricted environment

4. Conclusion and Future Work

4.1 Conclusion

4.1.1 Our contribution

4.1.2 What student can learn

4.2 Future work

5. How to Reproduce Our Work

5.1 Step 1“How to record a voice signal?”

5.2 Step 2 “How to convert audio files to FFT and build FFT based DB?”

5.3 Step 3 “How to convert audio files to spectrogram and build spectrogram based DB?”

5.4 Step 4 “How to train Inception V3 model?”

5.5 Step 5 “How to perform real-time demonstration?**”**

**Chapter 1**

**Introduction**

In this chapter, a wide background about deep learning and its famous applications will be introduced. To solve classification problem in deep learning there are a bunch of strong algorithms, which shows a small error margins, these algorithms will appear in this chapter, as one of them will be the heart of our work.

**1.1 Deep Learning background**

The umbrella under which both Machine Learning (ML) and Deep Learning (DL) lives is Artificial Intelligence (AI). Now-a-days, AI is so much widespread that all the current and future developments are dependent on it. In other words, AI is the broad concept of machines being able to carry out tasks in modern era.

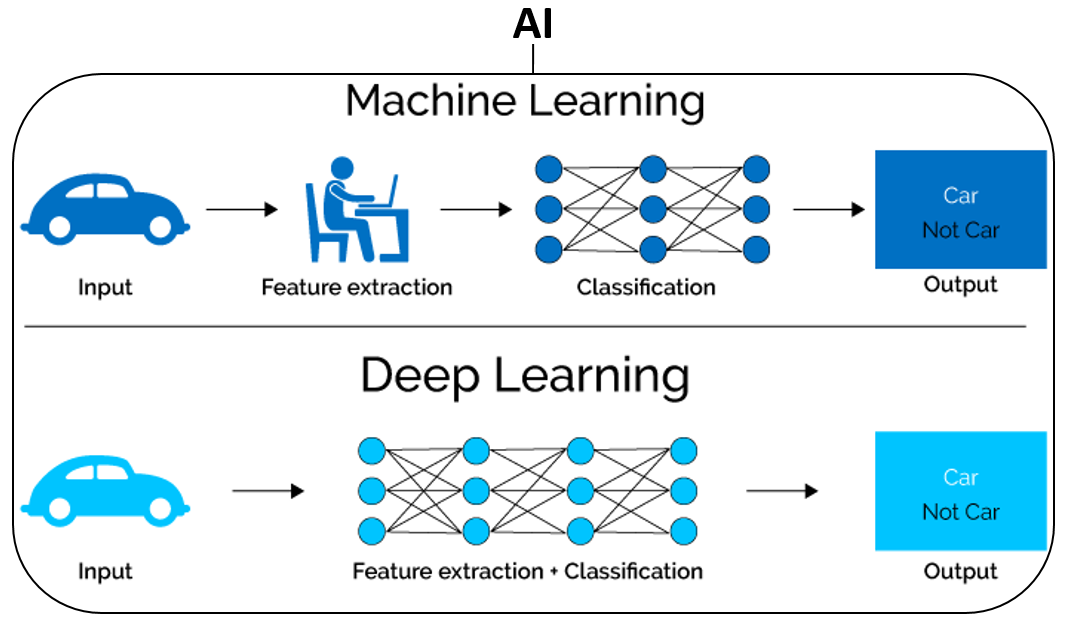


Figure 1.1. AI (Machine Learning vs Deep Learning) [1].

Traditional ML approaches worked like the top half of the figure 1.1. We would have to design a feature extraction algorithm which generally involved a lot of heavy mathematics (complex design), was not very efficient, and did not perform too well at all (accuracy level just was not suitable for real-world applications). After doing all of that, we would also have to design a whole classification model to classify your input given the extracted features.

With deep networks “DL”, we can perform feature extraction and classification in one shot as shown in bottom half of figure 1.1, which means we only have to design one model. DL uses a subset of ML techniques and tools, helping solve any issue that requires "thought," whether it is human or artificial. DL is popular right now because it is easy and it works.

Most modern DL models are based on an Artificial Neural Network (ANN) became more powerful, complex and literally deeper with many layers and neurons. The "deep" in "deep learning" refers to the number of layers through which the data is transformed, where each level of the network learns to transform its input data into a slightly more abstract and composite representation [2].

**1.1.1 Image Classification**

In 2009, professor and head of the Artificial Intelligence Lab at Stanford University, Fei-Fei Li launched ImageNet, it’s a very large and free database of more than 14 million labeled images available to researchers, educators, and students. This huge amount of data opens the door in front of training neural nets [3].

A lot of training models and algorithms have been developed since this time to use this open source labeled database trying to solve the classification problem with smallest error values. Since 2010, the annual ImageNet Large Scale Visual Recognition Challenge (ILSVRC) is a competition where research teams evaluate their algorithms on the given data set and compete to achieve higher accuracy on several visual recognition tasks [4], see figure 1.2.

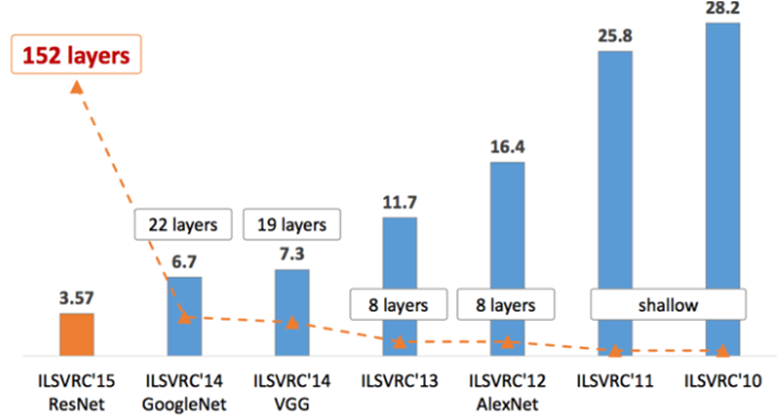


Figure 1.2. ImageNet-Large Scale Visual Recognition Challenge, showing winners,

number of used layers and achieved accuracy [5].

Image classification is one of machine learning application; there are many applications used in different fields like, speech recognition, medical diagnosis, statistical arbitrage, Learning Associations etc. In speech recognition, a software application recognizes spoken words and translation it into text. For medical diagnosis, machine learning can prediction of disease progression, for the extraction of medical knowledge for outcomes research, for therapy planning and support, and for overall patient management. In finance, statistical arbitrage refers to automated trading strategies that are typical of a short term and involve a large number of securities. In such strategies, the user tries to implement a trading algorithm for a set of securities based on quantities such as historical correlations and general economic variables. Learning association is the process of developing insights into various associations between products. A good example is how seemingly unrelated products may reveal an association to one another. When analyzed in relation to buying behaviors of customers [6].

**1.1.1.1  CNN**

Rosenblatt researched the artificial neural networks as early in 1960s; it was only in late 2000s when deep learning using neural networks took off. The key enabler was the scale of computation power and datasets with Google pioneering research into deep learning. In July 2012, researchers at Google exposed an advanced neural network to a series of unlabeled, static images sliced from YouTube videos. To their surprise, they discovered that the neural network learned a cat-detecting neuron on its own, supporting the popular assertion that “the internet is made of cats”. One of the neurons in the artificial neural network, trained from still frames from unlabeled YouTube videos, learned to detect cats, as displayed in figure 1.3.



Figure 1.3. Image of cat from Google’s blog [7].

The technique that Google researchers used is called Convolutional Neural Networks (CNN), a type of advanced artificial neural network. Convolution is a process where the network tries to label the input signal by referring to what it has learned in the past. Neural networks, as its name suggests, is a machine learning technique which is modeled after the brain structure. It comprises of a network of learning units called neurons. These neurons learn how to convert input signals (e.g. picture of a cat) into corresponding output signals (e.g. the label “cat”), forming the basis of automated recognition.

Let us take the example of automatic image recognition. The process of determining whether a picture contains a cat involves an activation function. If the picture resembles prior cat images the neurons have seen before, the label “cat” would be activated. Hence, the more labelled images the neurons are exposed to, the better it learns how to recognize other unlabeled images. We call this the process of training neurons as described in figure 1.4.

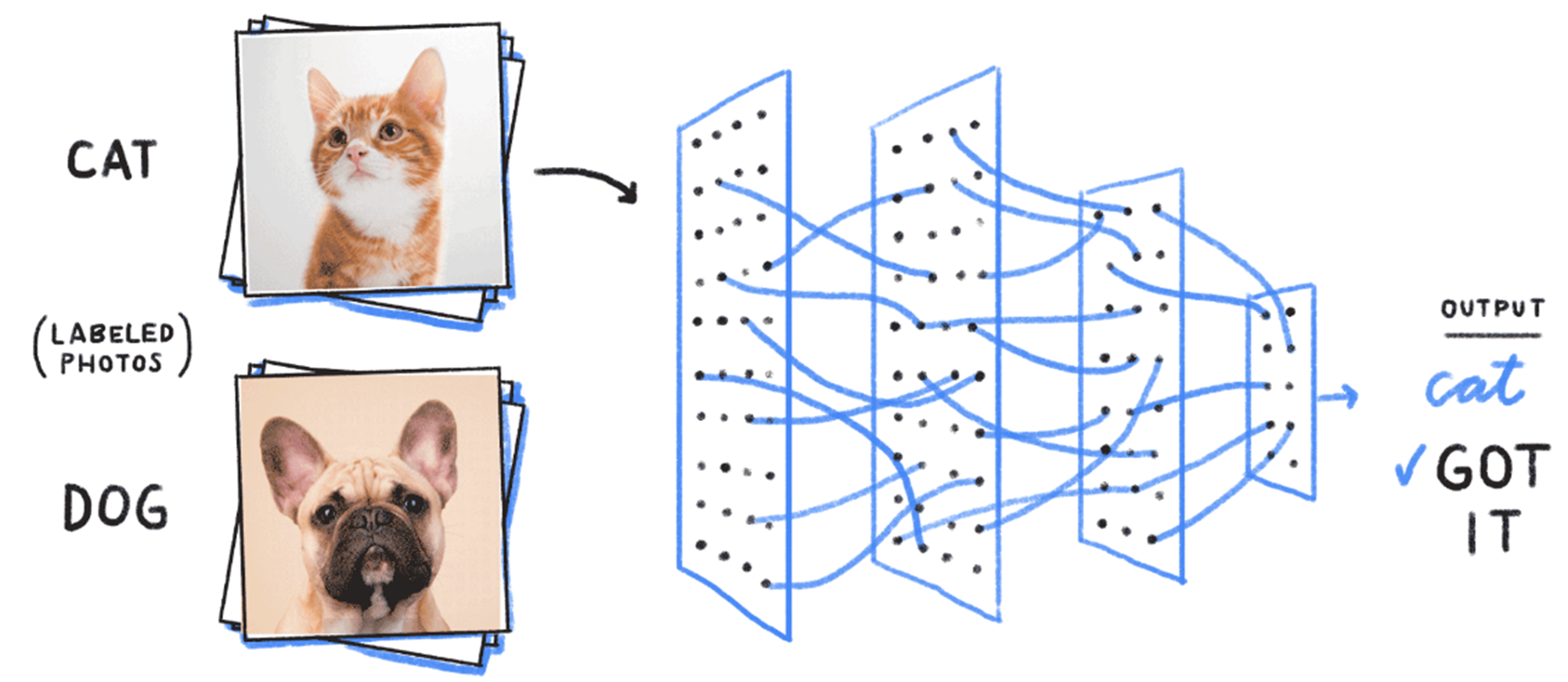


Figure 1.4. Image classification using CNN [8].

The process of building a CNN always involves five major layers as shown in figure 1.5:

Layer – 1: Input

Layer – 2: Convolutional

Layer – 3: Pooling

Layer – 4: Fully connected

Layer – 5: Output

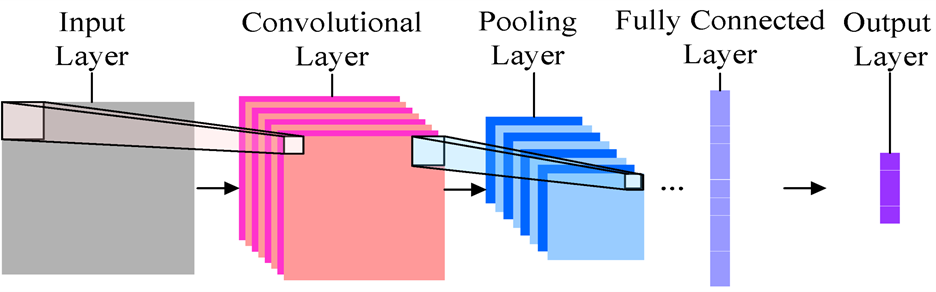


Figure 1.5. CNN layers [9].

**1.1.1.2 Inception V3**

Inception V3 is a widely-used image recognition model. It gains popularity after wining the ImageNet 2014 competition. During this competition, Inception v3 exhibits the error rate of 6.67% by having 22 layer deep CNN architecture. It is a widely-used image recognition model.

The goal of the inception module is to act as a “multi-level feature extractor” by computing 1×1, 3×3, and 5×5 convolutions within the same module of the network, the output of these filters are then stacked along the channel dimension and before being fed into the next layer in the network, as shown in figure 1.6. The original incarnation of this architecture was called GoogleNet, but subsequent manifestations have simply been called Inception V*n* where *n* refers to the version number put out by Google [10], [11].

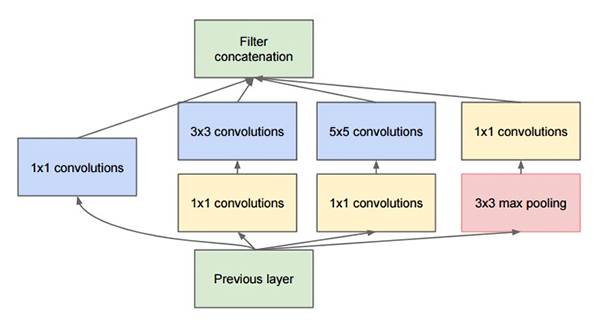


Figure 1.6. The original Inception module used in GoogleNet [10].

The model itself is made up of symmetric and asymmetric building blocks, including convolutions, average pooling, max pooling, concats, dropouts, and fully connected layers, as shown in figure 1.7.

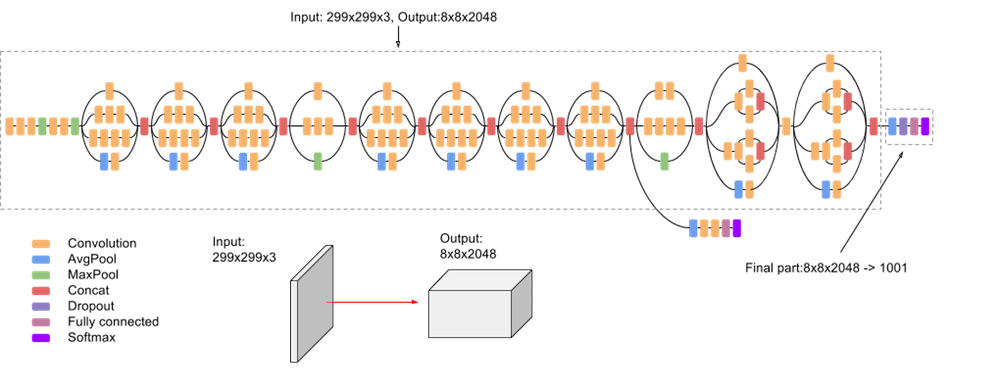


Figure 1.7. Inception V3 layers [12].

**1.1.2 Voice Recognition**

Speech recognition is one of the famous application in deep learning field. Where voice signal is an input and text generation is an output. The core idea of this application is how to interface and analyze each word to search for text matching to identify the written word/sentence.

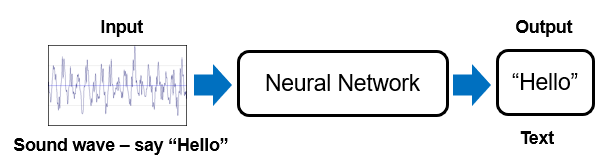


Figure 1.8. Speech recognition concept.

The main problem with the speech recognition is that, people can pronounce the same word in different speeds like “Hello” and “Hellllooo”, in both cases the audio file should be recognized as the same text. To solve such problems, additional processing steps should be used.

In our project, the deep learning model is not designed to search for a text matching, instead, it searches for speaker identity through feature matching in the transformed image either FFT or spectrogram.

In general, both systems use the voice signals as input but the way to train the model is totally different, in next chapters a detailed explanation about how to interface, train the model, evaluate the result, enhance the accuracy and step by step implementation procedure will be discussed.

**Chapter 2**

**Proposed System**

In this chapter an overall description for our proposed “voice classifier” system will be introduce, including an explanation for system’s parts. This chapter is strongly connected to chapter five, where the reader can find how to practically reproduce this work.

**2.1 Environment setup**

Linux “Ubuntu” used as an operating system, where Tensorflow installed as a machine learning platform. Python is the programming language which we used for coding voice interface, voice conversion and demonstration interface. Getting help from GitHub which is commonly used to host open-source software projects.

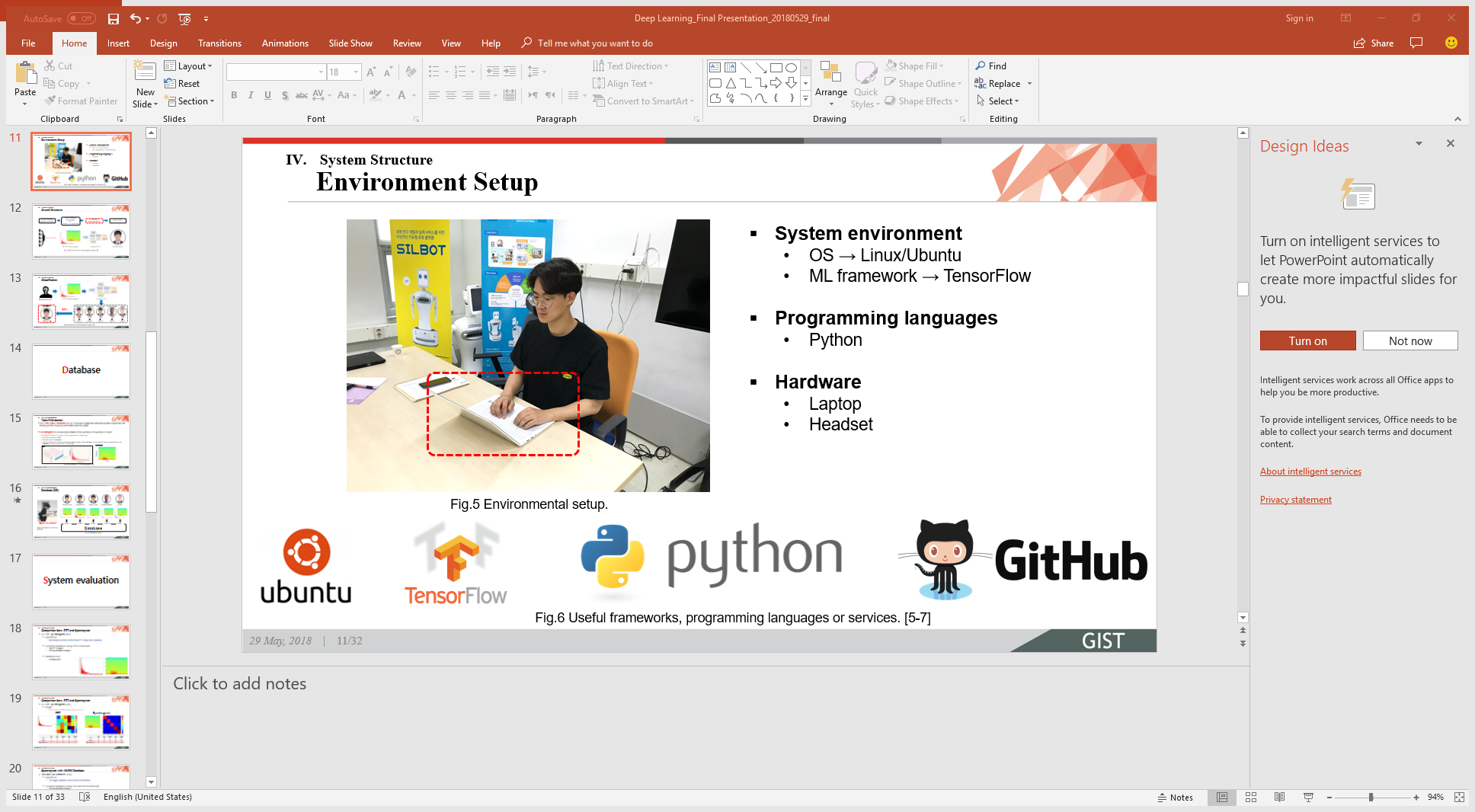


Figure 2.1. Operating system, ML framework,   
programming language and open sources.

**2.2 Overall structure**

Our system has four main stages for both training and testing. Starting with recording the audio signals followed by converting the audio signals to Fast Fourier Transform (FFT) or spectrogram in the form of images. Then use these images as an input to the classifier, which is inception V3, in our case. The fourth stage is recognition of speaker using the output result of the classifier according to the prelabeled DB.

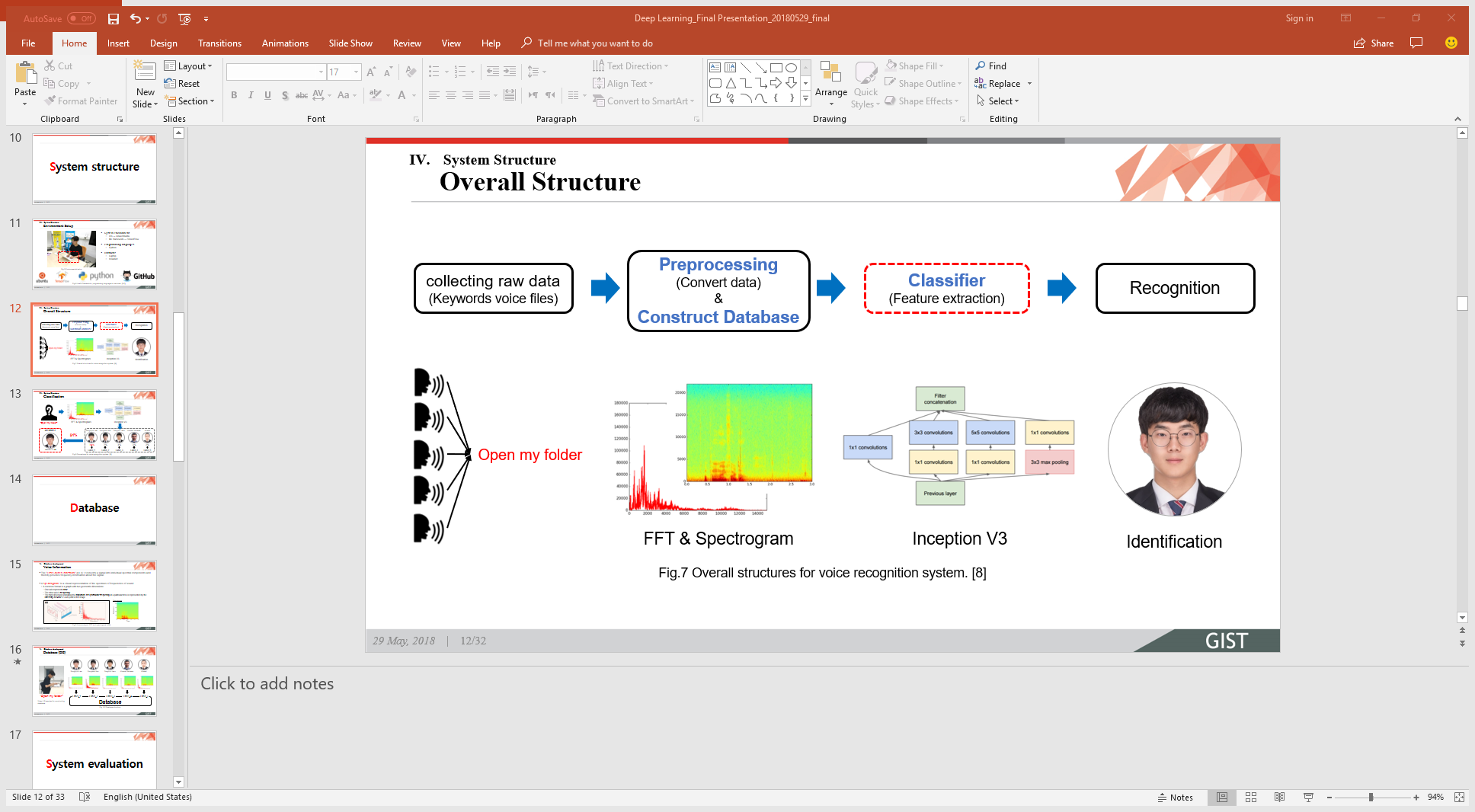


Figure 2.2. Overall structures for voice recognition system

**2.2.1 Voice signal interface and conversion**

**2.2.1.1 Voice signal interface**

Our main goal is to classify the speaker voice to know his identity. Recording the audio signal of the speaker to build our database is essential and primary step in our implementation. LG gram core i7 laptop with 16 GB RAM, aside with Sony headset “ZX110AP” as a voice input hardware with cool feature like noise cancellation are used. For the software interface, an open source code which record the speaker voice for 3 seconds is used, See (chapter 5 – step1).



Figure 2.3. Used hardware

**2.2.1.2 Voice signal conversion**

Audio signal is the input to our model, but is it proper to use the audio signal as an input for Inception V3 model? As mentioned previously that Inception V3 shows high accuracy for image classification, so using the audio signal in its original form is not proper for Inception V3 model. So that, a conversion from audio form to image form is required. Converting the audio signal to its corresponding FFT or spectrogram and save them in image form is the used technique.

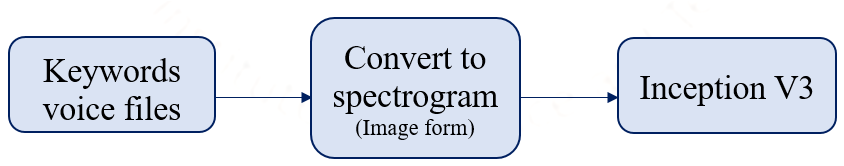


Figure 2.4. Prepare input data for inception V3.

To know how to convert audio signal to the corresponding FFT form or spectrogram form and save it as an image see (chapter5- step2 & step3).

**2.2.2 Database (DB)**

As described in the previous section a conversion for audio signal to FFT and spectrogram forms was important as a preprocessing step to fit our data to Inception V3 input. Therefore, it is clear that FFT and spectrogram are the main components for voice classifier system’s DB. Before describing how database is collected, it is important to know more about the general properties of FFT and spectrogram.

* The FFT (Fast Fourier Transform): It converts a signal into individual spectral components and thereby provides frequency information about the signal.
* A spectrogram: is a visual representation of the spectrum of frequencies of sound
* A common format is a graph with three geometric dimensions:
* One axis represents time
* The other axis is frequency
* The third dimension indicating the amplitude of a particular frequency at a particular time is represented by the intensity or color of each point in the image.

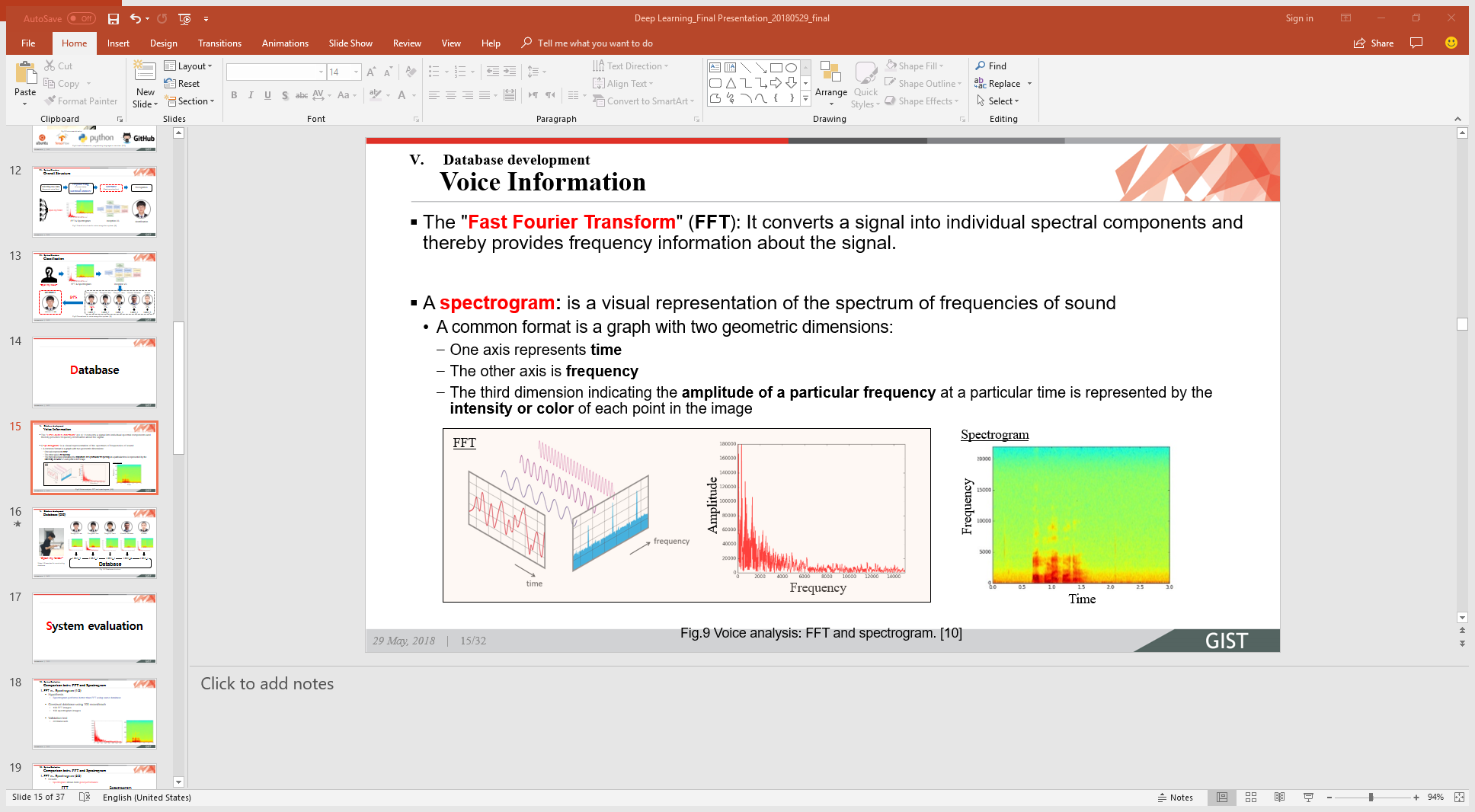


Figure 2.5. FFT and spectrogram.

This is how database was constructed, five team members speaking a sentence “Open my folder” for hundreds of times then the program stores the labeled FFT and spectrogram databases for each user. To know how to construct databases please see (Chapter 5 – step 2 & step3) as they are repeated frequently with any input voice signal.

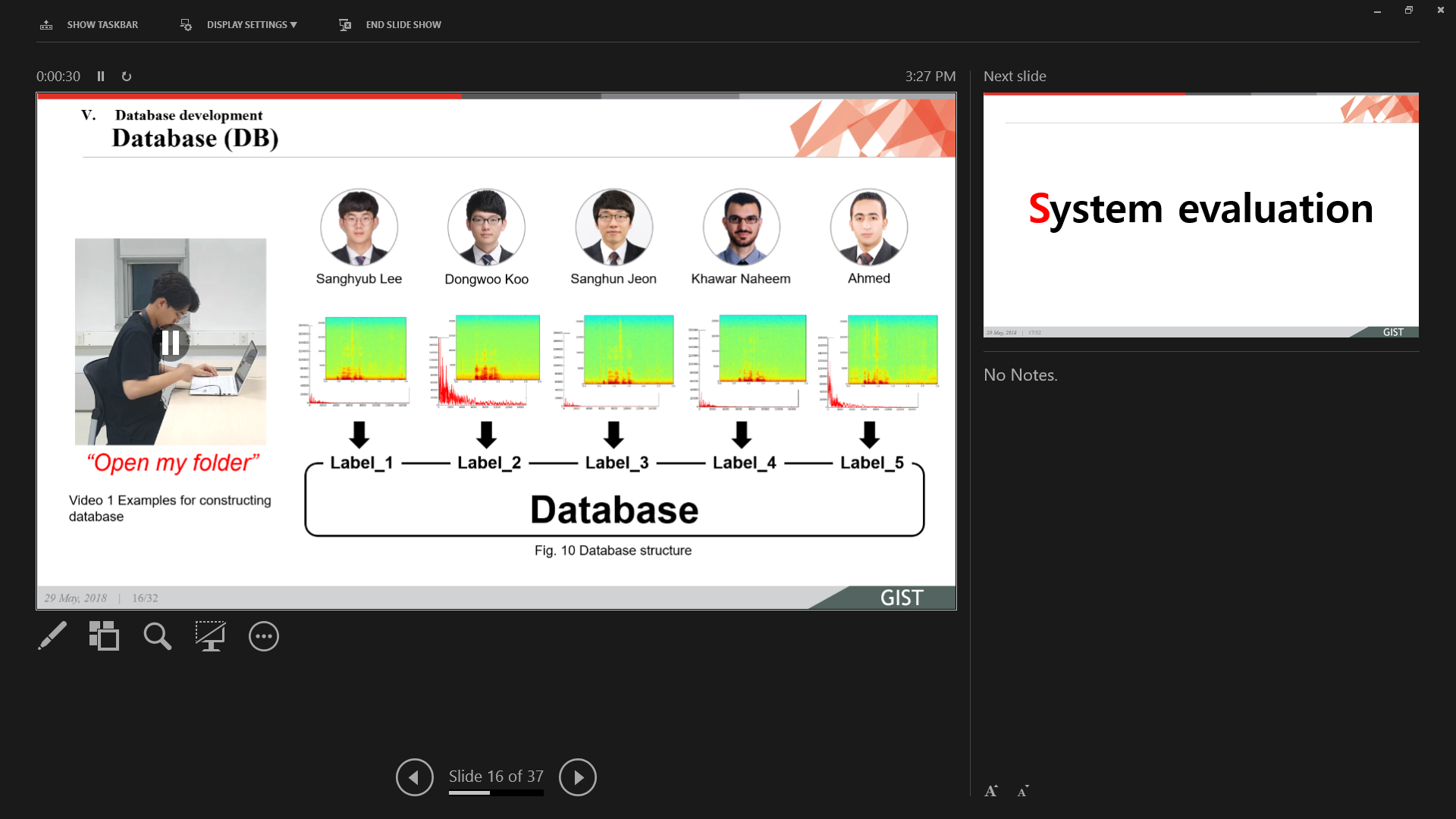


Figure 2.6. Database structure.

**2.2.2 Main algorithm**

After training the Inception V3 model with the customize database (Chapter 5 – step 4), the model will be able to classify the input voice by showing the probability as percentage for each predefined “Registered” users or team members. Therefore, user with high probability, seems to be the speaker.

Each registered user get a probability value based on how much the features on his “Real-time” FFT/spectrogram match the prestored DB.

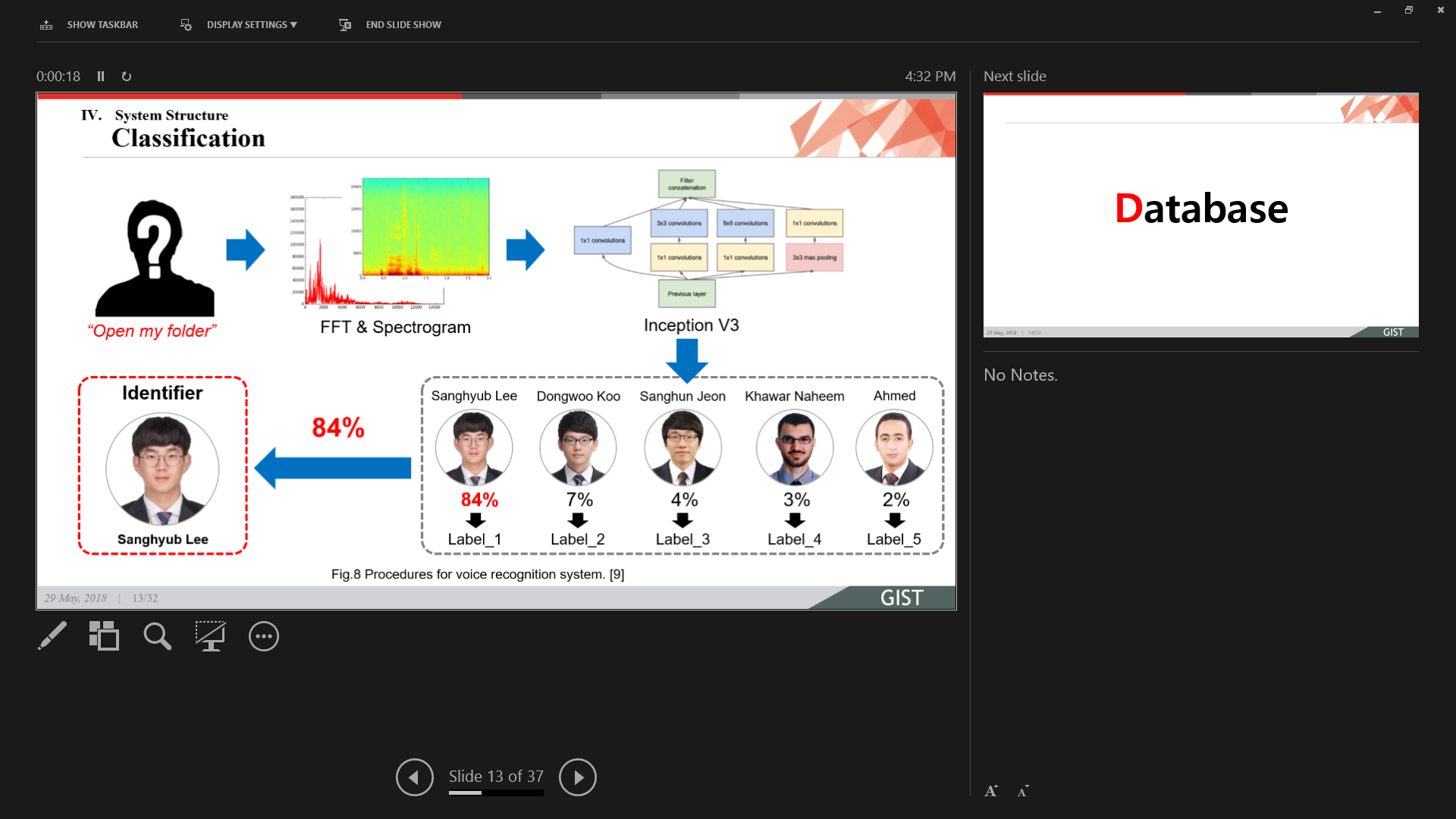


Figure 2.7. Procedure for voice recognition system.

A demonstration program has been designed to show how our system is working, in this program unknown speaker “From the registered user” asked to speak through the microphone, then based on the assigned probability by the model for each registered user, the program shows the user’s picture for whom the Inception V3 model get the higher probability. See how to perform real-time demonstration (Chapter 5 – step5).

**Chapter 3**

**System Evaluation**

In this chapter, the performance of the proposed system will be evaluated. Here, the performance is defined as the feature matching accuracy between the input team member’s voice and trained Inception V3 model which is trained using the in-house built DB. The evaluation is divided into three scenarios, 1. Comparison between FFT- and spectrogram-based DB, 2. Accuracy enhancement using bigger DB, 3. Testing accuracy enhancement under restricted environment. The confusion matrix is used to highlights and distinguishes the voice recognition/classification performance of our Inception V3 model for single speaker voice input. In addition, the confusion matrix concludes that how many trials are correctly matched with input speaker’s voice relative to the labeled DB.

**3.1 Scenarios**

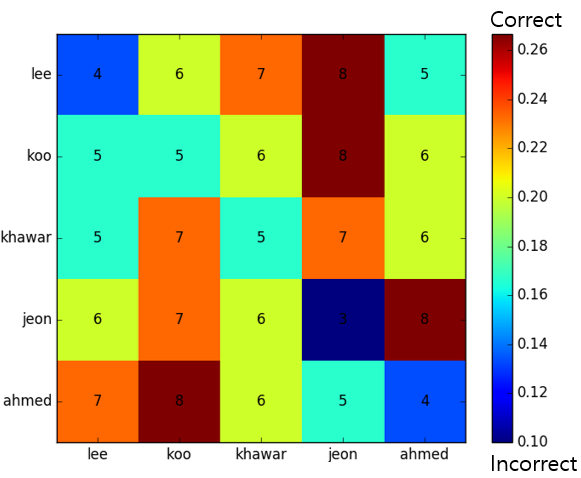
The detail of three scenarios used for performance evaluation are as under:

**3.1.1 Scenario1:** **Comparison between FFT- and spectrogram-based DB**

The purpose of this scenario is to select the best conversion method, between FFT and spectrogram, which transforms input voice into image.

In this scenario, the hypothesis1 (H1) is made. The DB is constructed using 100 FFT images and 100 spectrogram images for team member’s voice input. After that, 30 trials for each team member’s voice input are performed to validate the desired performance.

**H1:** Spectrogram performs better than FFT using the same size of DB.



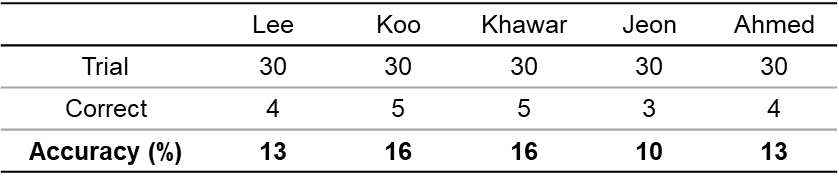
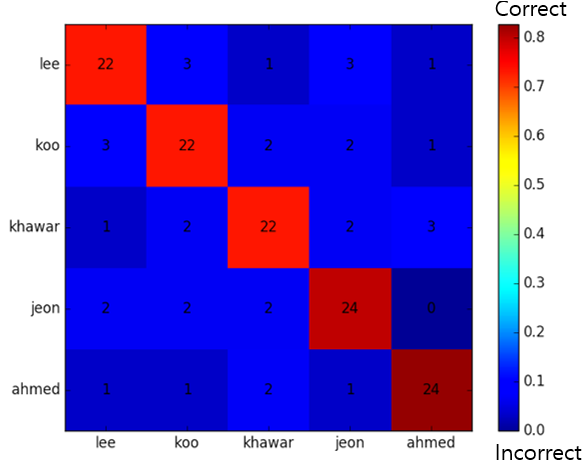


Figure 3.1. Scenario1: Confusion matrix graph and accuracy table using FFT based DB.



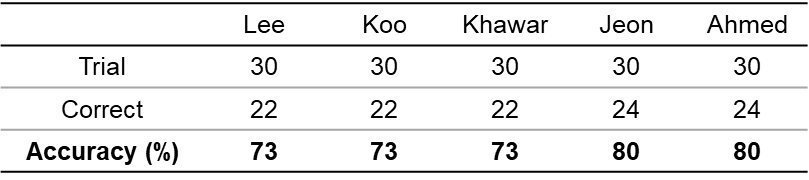


Figure 3.2. Scenario1: Confusion matrix graph and accuracy table

using spectrogram based DB.

**Result:** The FFT based DB and spectrogram based DB achieve the accuracy of 13% (on average) and 76% (on average), respectively, which confirms H1, as described in figures 3.1 and 3.2. Hence, it is confirmed from scenario1 that spectrogram based DB exhibits the high accuracy than the FFT based DB. So, from now on, only spectrogram based DB will be used for evaluation.

**3.1.2 Scenario2:** **Accuracy enhancement using bigger DB**

The purpose of this scenario is to improve the accuracy by increasing the size of spectrogram based DB. In this scenario, the hypothesis2 (H2) is made. The DB is constructed using 100 and 500 spectrogram images for each team member’s voice input. After that, 30 trials for each team member’s voice input are executed to validate the desired performance.

**H2:** The bigger DB enhances the model accuracy.

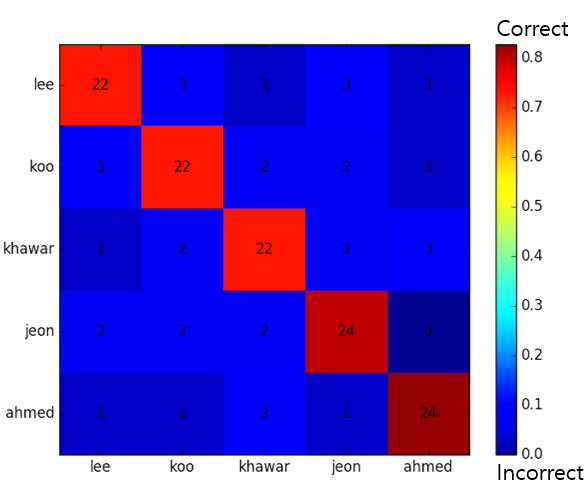
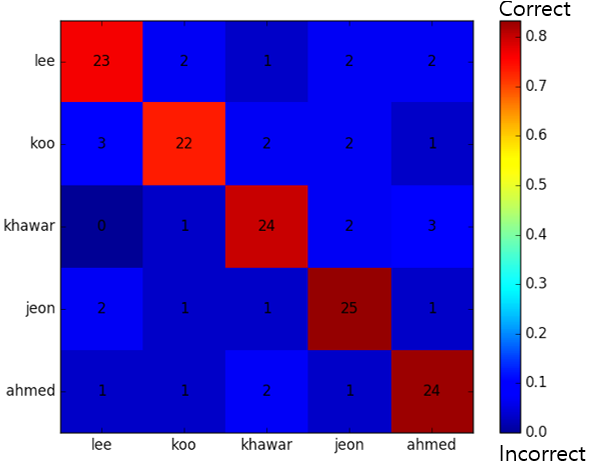




Figure 3.3. Scenario2: Confusion matrix graph and accuracy table

using spectrogram based DB of size 100.



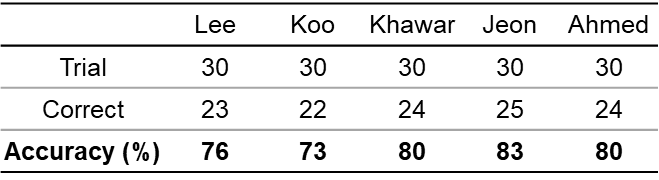


Figure 3.4. Scenario2: Confusion matrix graph and accuracy table

using spectrogram based DB of size 500.

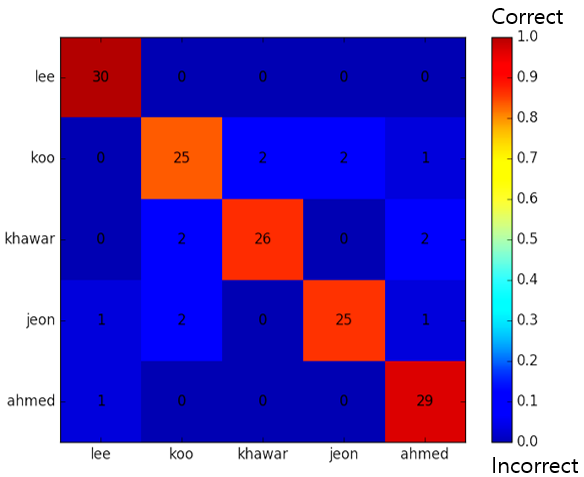
**Result:** The bigger DB does not significantly affect the accuracy, which actually contradicts H2, as displayed in figures 3.3 and 3.4. The reason of this contradiction is that when all team members record their voices under flexible environment (different sitting place/room and different microphone hardware). So according to this reason, the next scenario evaluates the accuracy under restricted environment (similar sitting place/room and similar microphone hardware).

**3.1.3 Scenario3: Testing accuracy enhancement under restricted environment**

The purpose of this scenario is to improve the accuracy under restricted environment. Furthermore, the used microphone hardware has a noise cancellation feature.

In this scenario, the hypothesis3 (H3) is made. The DB is constructed using 100 spectrogram images for each team member’s voice input under restricted environment. After that, 30 trials for each team member’s voice input are tested to validate the desired performance.

**H3:** The restricted environment and noise cancellation feature of microphone hardware significantly enhance the accuracy.



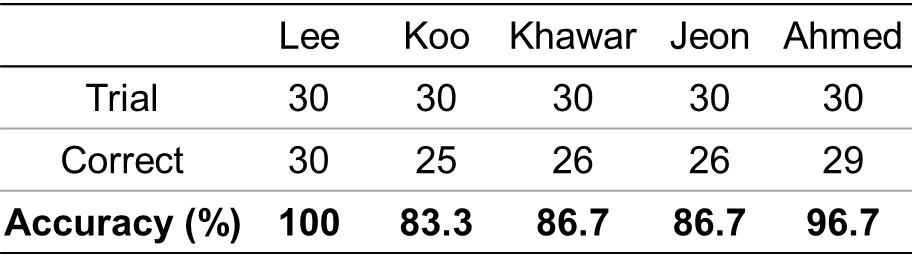


Figure 3.5. Scenario3: Confusion matrix graph and accuracy table

using spectrogram based DB under restricted environment.

**Result:** Under the restricted environment, the significant improvement in accuracy of 95% (on average) is accomplished which verifies H3, as shown in figure 3.5.

**Chapter 4**

**Conclusion and Future work**

In this chapter, the conclusion of all the chapters as well as the scope of the proposed system into the real-life applications will be described.

**4.1 Conclusion**

* Chapter1 elaborates the introductory background necessary for the proposed system.
* Chapter2 explains the overall structure of proposed system from beginning to the end point.
* A deep learning based voice recognition system is developed for a single speaker using a sentence “open my folder”.
* A DB of size (5\*100=500) is built using a sentence “open my folder” in terms of spectrogram images.
* Chapter3 validates the performance of proposed system by considering the three different scenarios.
* Average accuracy of 90.68 % is achieved using the spectrogram based DB under restricted environment compared with the FFT average accuracy of 13.6 %. Hence, the spectrogram is recommended conversion method to transforms input voice into image.
* The restricted environment and noise cancellation feature of microphone hardware significantly enhance the accuracy.

**4.1.1 Our contribution**

* + - Developed our own “Database”
    - Interfaced the voice files to transform into images as “spectrogram”
    - Applied the open sourced state-of-the-art classifier “Inspection V3”
    - Wrote a good “tutorial” to whom it may concern (especially students)

**4.1.2 What students can learn**

* + - How to record Audio file (.wav)?
    - How to build a database from scratch?
    - How to transform Audio signals into images?
    - How to apply a deep learning algorithm and classify the input data?
    - How to evaluate and enhance the performance of a deep learning model?

**4.2 Future work**

* One shot learning and quick DB’s registration strategy should be adopt
* Improve the recognition accuracy
* Remove the environment dependency
* Shape a good application like:
* Hands-free (no touch) and eye-free (no visual) utilities for disabled peoples
* Bio-metric security

**Chapter 5**

**How to Reproduce Our Work**

This chapter is very important to anyone who want to reproduce or extend our work. A step by step explanation will be provided to save the time of the reader and take him directly to the point “How to”.

The following prerequisite software/packages are required:

* Ubuntu and TensorFlow should be installed in his/her pc
* He/she must have little expertise to execute programs from Linux terminal

**5.1 Step 1 “How to record a voice signal?”**

Run “voice\_recorder.py [13]” program in Linux terminal and start recording for 3 second for each voice track.



# voice\_recorder.py

#! /usr/bin/env python

# -\*- coding: utf-8 -\*-

# opensource URL: https://gist.github.com/sloria/5693955

**import** pyaudio

**import** wave

**import** time

**class** **Recorder(**object**):**

"""A recorder class for recording audio to a WAV file.

recode type: mono - channels=1 (default)

recode type: stereo - channels=2"""

**def** \_\_init\_\_**(**self**,** channels**=**1**,** rate**=**44100**,** frames\_per\_buffer**=**1024**):**

self**.**channels **=** channels

self**.**rate **=** rate

self**.**frames\_per\_buffer **=** frames\_per\_buffer

**def** open**(**self**,** fname**,** mode**=**'wb'**):**

**return** RecordingFile**(**fname**,** mode**,** self**.**channels**,** self**.**rate**,**

self**.**frames\_per\_buffer**)**

**class** **RecordingFile(**object**):**

"""Get stream audio and save as file"""

**def** \_\_init\_\_**(**self**,** fname**,** mode**,** channels**,**

rate**,** frames\_per\_buffer**):**

self**.**fname **=** fname

self**.**mode **=** mode

self**.**channels **=** channels

self**.**rate **=** rate

self**.**frames\_per\_buffer **=** frames\_per\_buffer

self**.**\_pa **=** pyaudio**.**PyAudio**()**

self**.**wavefile **=** self**.**\_prepare\_file**(**self**.**fname**,** self**.**mode**)**

self**.**\_stream **=** **None**

**def** \_\_enter\_\_**(**self**):**

**return** self

**def** \_\_exit\_\_**(**self**,** exception**,** value**,** traceback**):**

self**.**close**()**

**def** record**(**self**,** duration**):**

self**.**\_stream **=** self**.**\_pa**.**open**(**format**=**pyaudio**.**paInt16**,**

channels**=**self**.**channels**,**

rate**=**self**.**rate**,**

input**=True,**

frames\_per\_buffer**=**self**.**frames\_per\_buffer**)**

**for** \_ **in** range**(**int**(**self**.**rate **/** self**.**frames\_per\_buffer **\*** duration**)):**

audio **=** self**.**\_stream**.**read**(**self**.**frames\_per\_buffer**)**

self**.**wavefile**.**writeframes**(**audio**)**

**return** **None**

**def** stop\_recording**(**self**):**

self**.**\_stream**.**stop\_stream**()**

return self

def get\_callback(self):

def callback(in\_data, frame\_count, time\_info, status):

self.wavefile.writeframes(in\_data)

return in\_data, pyaudio.paContinue

return callback

def close(self):

self.\_stream.close()

self.\_pa.terminate()

self.wavefile.close()

def \_prepare\_file(self, fname, mode='wb'):

wavefile = wave.open(fname, mode)

wavefile.setnchannels(self.channels)

wavefile.setsampwidth(self.\_pa.get\_sample\_size(pyaudio.paInt16))

wavefile.setframerate(self.rate)

return wavefile

if \_\_name\_\_ == '\_\_main\_\_':

print('Voice Recoder Start!')

rec = Recorder(channels=2)

# for loop recodnig

for i in range(1, 101, 1):

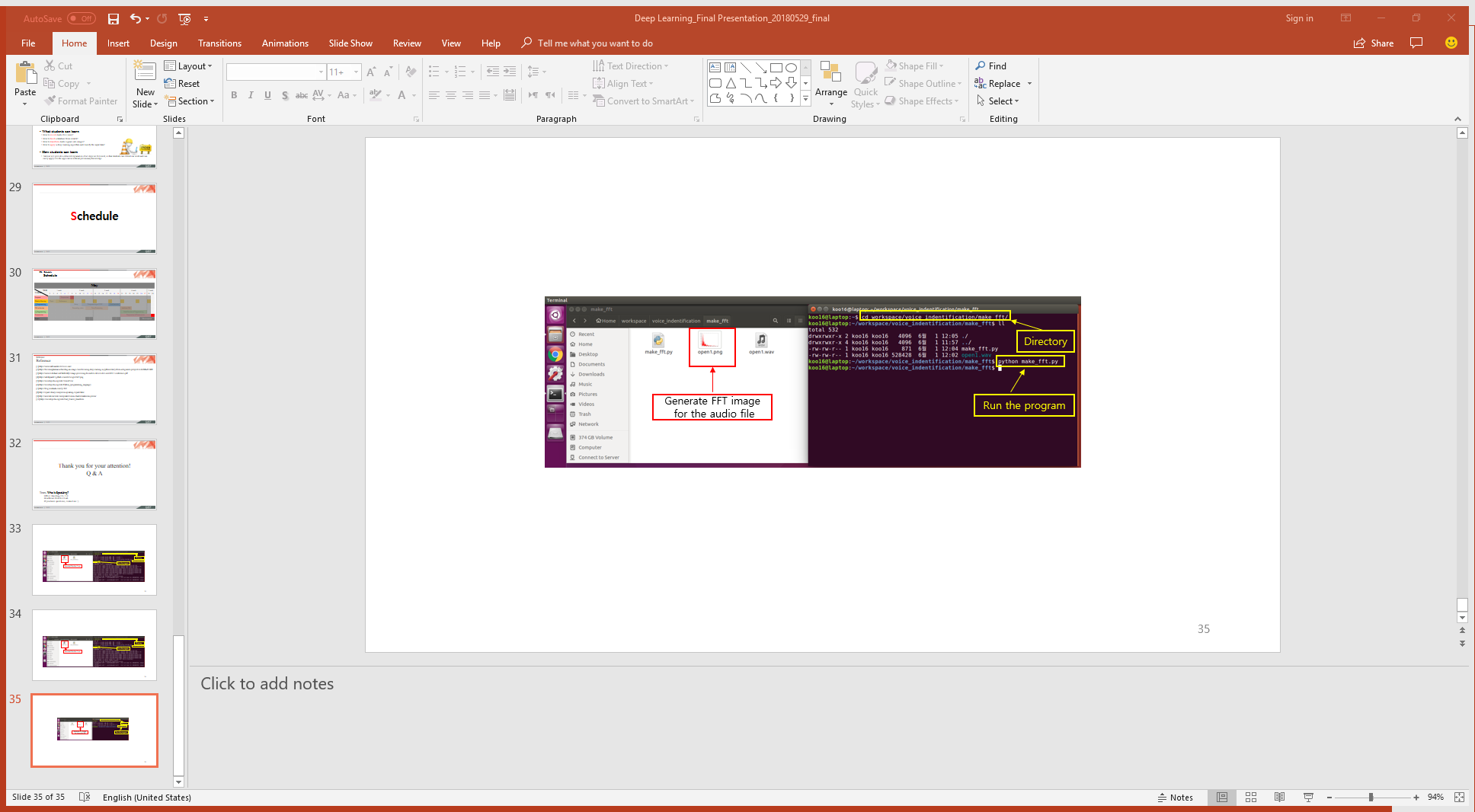
print('########## count: {} ##########'.format(i))

with rec.open('./open{}.wav'.format(i), 'wb') as recfile:

recfile.record(duration=3.0)

**5.2 Step 2 “How to convert audio files to FFT and build FFT based DB?”**

Run “make\_fft.py [14]” program in Linux terminal to convert audio file to FFT form.



# make\_fft.py

#! /usr/bin/env python

# -\*- coding: utf-8 -\*-

# opensource URL: https://stackoverflow.com/questions/23377665/python-scipy-fft-wav-files

**import** matplotlib**.**pyplot **as** plt

**from** scipy**.**fftpack **import** fft

**from** scipy**.**io **import** wavfile # get the api

**for** i **in** range**(**1**,** 2**,** 1**):**

# from voice file, do frequency analysis

fs**,** data **=** wavfile**.**read**(**'./open{}.wav'**.**format**(**i**))** # load the data

a **=** data**.**T**[**0**]** # this is a two channel soundtrack, I get the first track

b **=** **[(**ele **/** 2 **\*\*** 8.**)** **\*** 2 **-** 1 **for** ele **in** a**]** # this is 8-bit track, b is now normalized on [-1,1)

c **=** fft**(**b**)** # calculate fourier transform (complex numbers list)

d **=** len**(**c**)** **/** 2 # you only need half of the fft list (real signal symmetry)

# make fft images

plt**.**plot**(**abs**(**c**[:(**d **-** 1**)]),** 'r'**)**

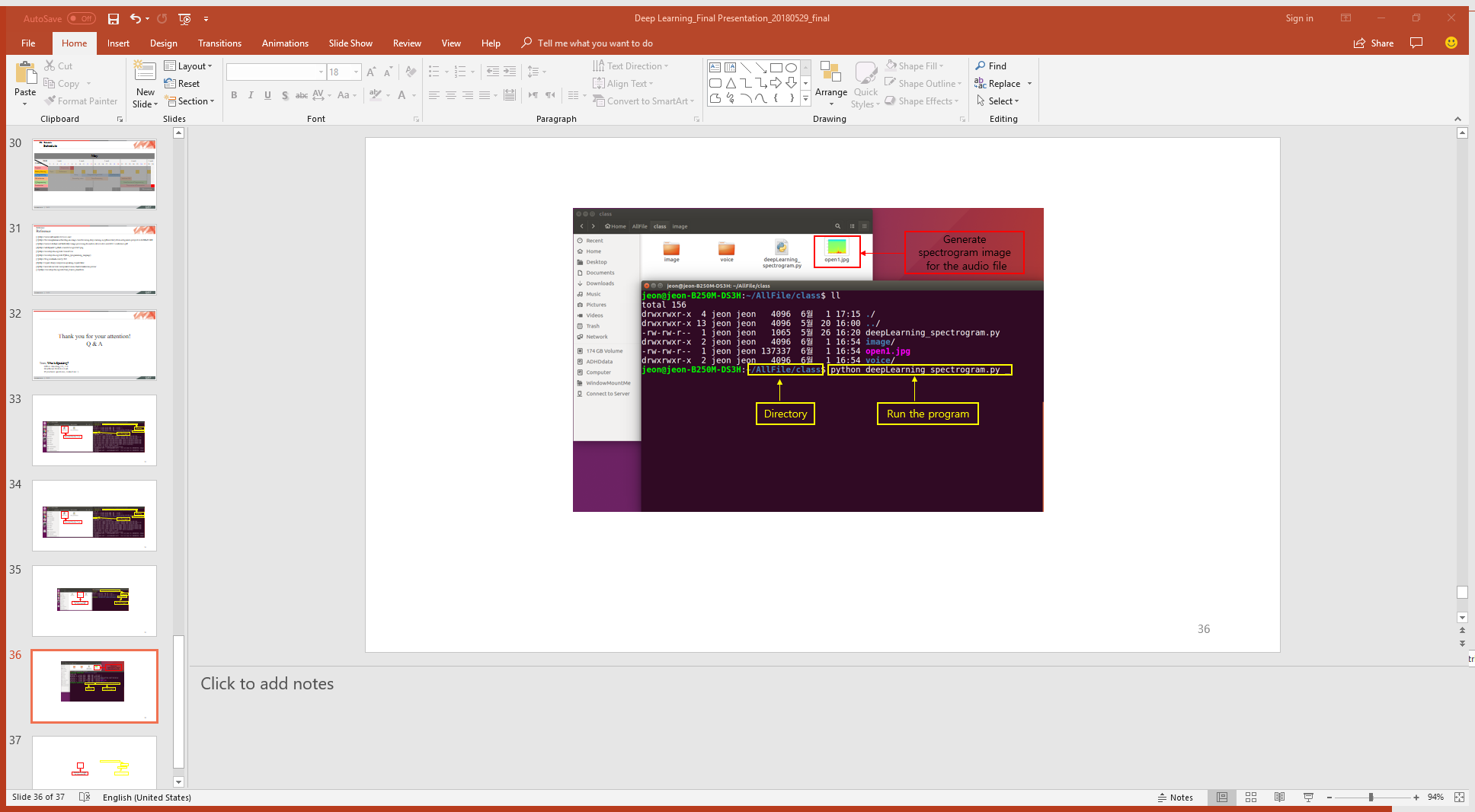
plt**.**axis**([**0**,** 15000**,** 0**,** 180000**])**

plt**.**savefig**(**'./open{}.png'**.**format**(**i**))**

plt**.**close**()**

**5.3 Step 3 “How to convert audio files to spectrogram and build spectrogram based DB?”**

Run “deeplearning\_spectrogram.py” program in Linux terminal to convert audio file to spectrogram form.

\

# deeplearning\_spectrogram.py

# !/usr/bin/env python

# -\*- coding:utf-8 -\*-

**import** os

**import** getpass

**import** matplotlib**.**pyplot **as** plt

**from** scipy **import** signal

**from** scipy**.**io **import** wavfil

user\_name **=** getpass**.**getuser**()** # User's Name of Computer

abspath\_path **=** '/home/' **+** str**(**user\_name**)** **+** '/AllFile/class/' # Abspath path of WAV

voice\_path **=** abspath\_path **+** 'voice/'

voice\_listdir **=** os**.**listdir**(**voice\_path**)** # Count of WAV files

listdir\_number **=** len**(**voice\_listdir**)** # Lists of WAV files

**for** number **in** range**(**listdir\_number**):**

wav\_file\_path **=** voice\_path **+** 'open' **+** str**(**number**+**1**)** **+** '.wav' # WAV files path

postprocessing\_image\_path **=** abspath\_path **+** '/image/' **+** 'open' **+** str**(**number**+**1**)** **+** '.jpg' # Save images files path

sample\_frequency**,** signalData **=** wavfile**.**read**(**wav\_file\_path**)** # Read WAV files

plt**.**specgram**(**signalData**[:,**0**],** Fs**=**sample\_frequency**)** # From WAV files to Spectrogram

plt**.**axis**([**0**,** 3**,** 0**,** 22000**])** # Figures

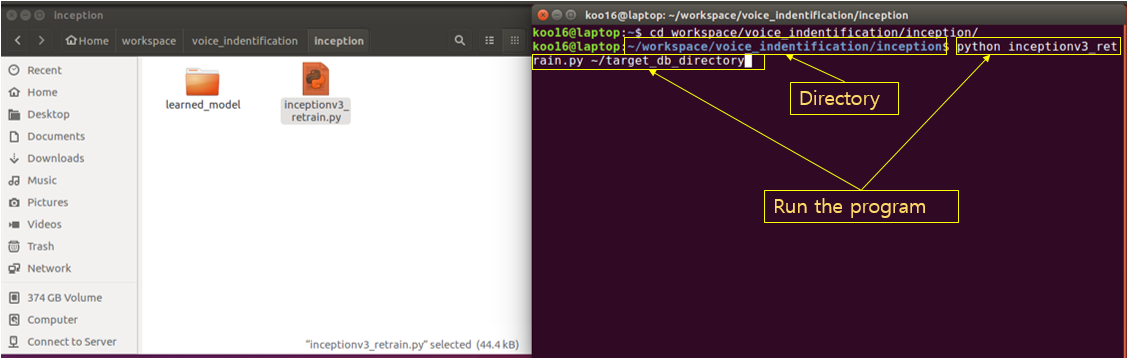
plt**.**savefig**(**postprocessing\_image\_path**)** # Save image files

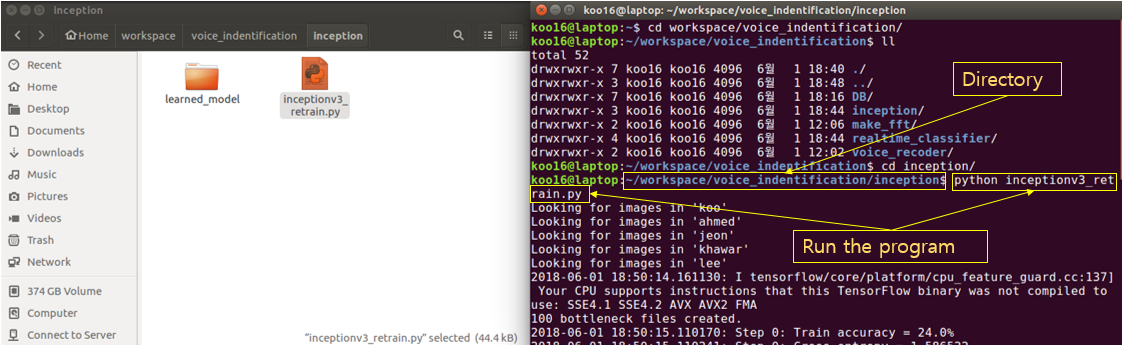
**5.4 Step** **4 “How to train Inception V3 model?”**

Run “inceptionv3\_retrain.py [15]” program in Linux terminal to train Inception V3 using already made DB.

The four training steps are as under:

1. The python script “inceptionv3\_retrain.py” will generate a number of labels which user wants to classify.
2. It will load all image files in each categories (directory) as specific labels.
3. The images will be gone through the image preprocessing such as resizing.
4. The Inception V3 will be trained with the processed images and their label.





# inceptionv3\_retrain.py

#! /usr/bin/env python

# -\*- coding: utf-8 -\*-

# opensource URL: http://solarisailab.com/archives/1422

"""Inception v3 architecture 모델을 이용한 간단한 Transfer Learning (TensorBoard 포함)

This example shows how to take a Inception v3 architecture model trained on

ImageNet images, and train a new top layer that can recognize other classes of

images.

The top layer receives as input a 2048-dimensional vector for each image. We

train a softmax layer on top of this representation. Assuming the softmax layer

contains N labels, this corresponds to learning N + 2048\*N model parameters

corresponding to the learned biases and weights.

Here's an example, which assumes you have a folder containing class-named

subfolders, each full of images for each label. The example folder flower\_photos

should have a structure like this:

~/flower\_photos/daisy/photo1.jpg

~/flower\_photos/daisy/photo2.jpg

...

~/flower\_photos/rose/anotherphoto77.jpg

...

~/flower\_photos/sunflower/somepicture.jpg

The subfolder names are important, since they define what label is applied to

each image, but the filenames themselves don't matter. Once your images are

prepared, you can run the training with a command like this:

```bash

bazel build tensorflow/examples/image\_retraining:retrain && \

bazel-bin/tensorflow/examples/image\_retraining/retrain \

--image\_dir ~/flower\_photos

```

Or, if you have a pip installation of tensorflow, `retrain.py` can be run

without bazel:

```bash

python tensorflow/examples/image\_retraining/retrain.py \

--image\_dir ~/flower\_photos

```

You can replace the image\_dir argument with any folder containing subfolders of

images. The label for each image is taken from the name of the subfolder it's

in.

This produces a new model file that can be loaded and run by any TensorFlow

program, for example the label\_image sample code.

To use with TensorBoard:

By default, this script will log summaries to /tmp/retrain\_logs directory

Visualize the summaries with this command:

tensorboard --logdir /tmp/retrain\_logs

"""

**from** \_\_future\_\_ **import** absolute\_import

**from** \_\_future\_\_ **import** division

**from** \_\_future\_\_ **import** print\_function

**import** argparse

**from** datetime **import** datetime

**import** hashlib

**import** os**.**path

**import** random

**import** re

**import** struct

**import** sys

**import** tarfile

**import** numpy **as** np

**from** six**.**moves **import** urllib

**import** tensorflow **as** tf

**from** tensorflow**.**python**.**framework **import** graph\_util

**from** tensorflow**.**python**.**framework **import** tensor\_shape

**from** tensorflow**.**python**.**platform **import** gfile

**from** tensorflow**.**python**.**util **import** compat

FLAGS **=** **None**

# 모든 파라미터들은 특정한 모델 architecture와 묶여(tied) 있다.

# 우리는 Inception v3를 사용할 것이다. 이는 tensor 이름이나 사이즈들을 포함하고 있다.

# 만약 당신이 이 스크립트를 다른 모델에 사용하고 싶다면,

# 당신이 사용하는 network를 반영하도록 tensor 이름이나 사이즈들을 변경해야만 할 것이다.

DATA\_URL = 'http://download.tensorflow.org/models/image/imagenet/inception-2015-12-05.tgz'

BOTTLENECK\_TENSOR\_NAME = 'pool\_3/\_reshape:0'

BOTTLENECK\_TENSOR\_SIZE = 2048

MODEL\_INPUT\_WIDTH = 299

MODEL\_INPUT\_HEIGHT = 299

MODEL\_INPUT\_DEPTH = 3

JPEG\_DATA\_TENSOR\_NAME = 'DecodeJpeg/contents:0'

RESIZED\_INPUT\_TENSOR\_NAME = 'ResizeBilinear:0'

MAX\_NUM\_IMAGES\_PER\_CLASS = 2 \*\* 27 - 1 # ~134M

def create\_image\_lists(image\_dir, testing\_percentage, validation\_percentage):

"""file system으로부터 training 이미지들의 list를 만든다.

이미지 디렉토리로부터 sub folder들을 분석하고, 그들을 training, testing, validation sets으로 나눈다.

그리고 각각의 label을 위한 이미지 list와 그들의 경로(path)를 나타내는 자료구조(data structure)를 반환한다.

인수들(Args):

image\_dir: 이미지들의 subfolder들을 포함한 folder의 String path.

testing\_percentage: 전체 이미지중 테스트를 위해 사용되는 비율을 나타내는 Integer.

validation\_percentage: 전체 이미지중 validation을 위해 사용되는 비율을 나타내는 Integer.

반환값들(Returns):

각각의 label subfolder를 위한 enrtry를 포함한 dictionary A dictionary

(각각의 label에서 이미지드릉ㄴ training, testing, validation sets으로 나뉘어져 있다.)

"""

if not gfile.Exists(image\_dir):

print("Image directory '" + image\_dir + "' not found.")

return None

result = {}

sub\_dirs = [x[0] for x in gfile.Walk(image\_dir)]

# root directory는 처음에 온다. 따라서 이를 skip한다.

is\_root\_dir = True

for sub\_dir in sub\_dirs:

if is\_root\_dir:

is\_root\_dir = False

continue

extensions = ['jpg', 'jpeg', 'JPG', 'JPEG', 'png', 'PNG']

file\_list = []

dir\_name = os.path.basename(sub\_dir)

if dir\_name == image\_dir:

continue

print("Looking for images in '" + dir\_name + "'")

for extension in extensions:

file\_glob = os.path.join(image\_dir, dir\_name, '\*.' + extension)

file\_list.extend(gfile.Glob(file\_glob))

if not file\_list:

print('No files found')

continue

if len(file\_list) < 20:

print('WARNING: Folder has less than 20 images, which may cause issues.')

elif len(file\_list) > MAX\_NUM\_IMAGES\_PER\_CLASS:

print('WARNING: Folder {} has more than {} images. Some images will '

'never be selected.'.format(dir\_name, MAX\_NUM\_IMAGES\_PER\_CLASS))

label\_name = re.sub(r'[^a-z0-9]+', ' ', dir\_name.lower())

training\_images = []

testing\_images = []

validation\_images = []

for file\_name in file\_list:

base\_name = os.path.basename(file\_name)

# 어떤 이미지로 리스트를 만들지 결정할때 파일 이름에 "\_nohash\_"가 포함되어 있으면 이를 무시할 수 있다.

# 이를 이용해서, 데이터셋을 만드는 사람은 서로 비슷한 사진들을 grouping할 수있다.

# 예를 들어, plant disease를 데이터셋을 만들기 위해서, 여러 장의 같은 잎사귀(leaf)를 grouping할 수 있다.

hash\_name = re.sub(r'\_nohash\_.\*$', '', file\_name)

# 이는 일종의 마법처럼 보일 수 있다. 하지만, 우리는 이 파일이 training sets로 갈지, testing sets로 갈지, validation sets로 갈지 결정해야만 한다.

# 그리고 우리는 더많은 파일들이 추가되더라도, 같은 set에 이미 존재하는 파일들이 유지되길 원한다.

# 그렇게 하기 위해서는, 우리는 파일 이름 그자체로부터 결정하는 안정적인 방법이 있어야만 한다.

# 따라서, 우리는 파일 이름을 hash하고, 이를 이를 할당하는데 사용하는 확률을 결정하는데 사용한다.

hash\_name\_hashed = hashlib.sha1(compat.as\_bytes(hash\_name)).hexdigest()

percentage\_hash = ((int(hash\_name\_hashed, 16) %

(MAX\_NUM\_IMAGES\_PER\_CLASS + 1)) \*

(100.0 / MAX\_NUM\_IMAGES\_PER\_CLASS))

if percentage\_hash < validation\_percentage:

validation\_images.append(base\_name)

elif percentage\_hash < (testing\_percentage + validation\_percentage):

testing\_images.append(base\_name)

else:

training\_images.append(base\_name)

result[label\_name] = {

'dir': dir\_name,

'training': training\_images,

'testing': testing\_images,

'validation': validation\_images,

}

return result

def get\_image\_path(image\_lists, label\_name, index, image\_dir, category):

""""주어진 index에 대한 이미지 경로(path)를 리턴한다.

인수들(Args):

image\_lists: 각각의 label에 대한 training image들의 Dictionary.

label\_name: 우리가 얻고자하는 이미지의 Label string.

index: 우리가 얻고자하는 이미지의 Int offset. 이는 레이블에 대한 가능한 이미지의 개수에 따라 moduloed 될 것이다.

따라서 임의의 큰값이 될 수도 있다.

image\_dir: training 이미지들의 subfolder들을 포함하고 있는 Root folder string

category: training, testing, 또는 validation sets으로부터 이미지에 pull할 Name string

반환값(Returns):

요청된 파라미터들이 만나게 될 이미지에 대한 파일 시스템 경로(file system path) string

"""

if label\_name not in image\_lists:

tf.logging.fatal('Label does not exist %s.', label\_name)

label\_lists = image\_lists[label\_name]

if category not in label\_lists:

tf.logging.fatal('Category does not exist %s.', category)

category\_list = label\_lists[category]

if not category\_list:

tf.logging.fatal('Label %s has no images in the category %s.',

label\_name, category)

mod\_index = index % len(category\_list)

base\_name = category\_list[mod\_index]

sub\_dir = label\_lists['dir']

full\_path = os.path.join(image\_dir, sub\_dir, base\_name)

return full\_path

def get\_bottleneck\_path(image\_lists, label\_name, index, bottleneck\_dir,

category):

""""Returns a path to a bottleneck file for a label at the given index.

Args:

image\_lists: Dictionary of training images for each label.

label\_name: Label string we want to get an image for.

index: Integer offset of the image we want. This will be moduloed by the

available number of images for the label, so it can be arbitrarily large.

bottleneck\_dir: Folder string holding cached files of bottleneck values.

category: Name string of set to pull images from - training, testing, or

validation.

Returns:

File system path string to an image that meets the requested parameters.

"""

return get\_image\_path(image\_lists, label\_name, index, bottleneck\_dir,

category) + '.txt'

def create\_inception\_graph():

""""Creates a graph from saved GraphDef file and returns a Graph object.

Returns:

Graph holding the trained Inception network, and various tensors we'll be

manipulating.

"""

with tf.Graph().as\_default() as graph:

model\_filename = os.path.join(

FLAGS.model\_dir, 'classify\_image\_graph\_def.pb')

with gfile.FastGFile(model\_filename, 'rb') as f:

graph\_def = tf.GraphDef()

graph\_def.ParseFromString(f.read())

bottleneck\_tensor, jpeg\_data\_tensor, resized\_input\_tensor = (

tf.import\_graph\_def(graph\_def, name='', return\_elements=[

BOTTLENECK\_TENSOR\_NAME, JPEG\_DATA\_TENSOR\_NAME,

RESIZED\_INPUT\_TENSOR\_NAME]))

return graph, bottleneck\_tensor, jpeg\_data\_tensor, resized\_input\_tensor

def run\_bottleneck\_on\_image(sess, image\_data, image\_data\_tensor,

bottleneck\_tensor):

"""Runs inference on an image to extract the 'bottleneck' summary layer.

Args:

sess: Current active TensorFlow Session.

image\_data: String of raw JPEG data.

image\_data\_tensor: Input data layer in the graph.

bottleneck\_tensor: Layer before the final softmax.

Returns:

Numpy array of bottleneck values.

"""

bottleneck\_values = sess.run(

bottleneck\_tensor,

{image\_data\_tensor: image\_data})

bottleneck\_values = np.squeeze(bottleneck\_values)

return bottleneck\_values

def maybe\_download\_and\_extract():

"""Download and extract model tar file.

If the pretrained model we're using doesn't already exist, this function

downloads it from the TensorFlow.org website and unpacks it into a directory.

"""

dest\_directory = FLAGS.model\_dir

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('Successfully downloaded', filename, statinfo.st\_size, 'bytes.')

tarfile.open(filepath, 'r:gz').extractall(dest\_directory)

def ensure\_dir\_exists(dir\_name):

"""Makes sure the folder exists on disk.

Args:

dir\_name: Path string to the folder we want to create.

"""

if not os.path.exists(dir\_name):

os.makedirs(dir\_name)

def write\_list\_of\_floats\_to\_file(list\_of\_floats, file\_path):

"""Writes a given list of floats to a binary file.

Args:

list\_of\_floats: List of floats we want to write to a file.

file\_path: Path to a file where list of floats will be stored.

"""

s = struct.pack('d' \* BOTTLENECK\_TENSOR\_SIZE, \*list\_of\_floats)

with open(file\_path, 'wb') as f:

f.write(s)

def read\_list\_of\_floats\_from\_file(file\_path):

"""Reads list of floats from a given file.

Args:

file\_path: Path to a file where list of floats was stored.

Returns:

Array of bottleneck values (list of floats).

"""

with open(file\_path, 'rb') as f:

s = struct.unpack('d' \* BOTTLENECK\_TENSOR\_SIZE, f.read())

return list(s)

bottleneck\_path\_2\_bottleneck\_values = {}

def create\_bottleneck\_file(bottleneck\_path, image\_lists, label\_name, index,

image\_dir, category, sess, jpeg\_data\_tensor,

bottleneck\_tensor):

"""Create a single bottleneck file."""

print('Creating bottleneck at ' + bottleneck\_path)

image\_path = get\_image\_path(image\_lists, label\_name, index,

image\_dir, category)

if not gfile.Exists(image\_path):

tf.logging.fatal('File does not exist %s', image\_path)

image\_data = gfile.FastGFile(image\_path, 'rb').read()

try:

bottleneck\_values = run\_bottleneck\_on\_image(

sess, image\_data, jpeg\_data\_tensor, bottleneck\_tensor)

except:

raise RuntimeError('Error during processing file %s' % image\_path)

bottleneck\_string = ','.join(str(x) for x in bottleneck\_values)

with open(bottleneck\_path, 'w') as bottleneck\_file:

bottleneck\_file.write(bottleneck\_string)

def get\_or\_create\_bottleneck(sess, image\_lists, label\_name, index, image\_dir,

category, bottleneck\_dir, jpeg\_data\_tensor,

bottleneck\_tensor):

"""Retrieves or calculates bottleneck values for an image.

If a cached version of the bottleneck data exists on-disk, return that,

otherwise calculate the data and save it to disk for future use.

Args:

sess: The current active TensorFlow Session.

image\_lists: Dictionary of training images for each label.

label\_name: Label string we want to get an image for.

index: Integer offset of the image we want. This will be modulo-ed by the

available number of images for the label, so it can be arbitrarily large.

image\_dir: Root folder string of the subfolders containing the training

images.

category: Name string of which set to pull images from - training, testing,

or validation.

bottleneck\_dir: Folder string holding cached files of bottleneck values.

jpeg\_data\_tensor: The tensor to feed loaded jpeg data into.

bottleneck\_tensor: The output tensor for the bottleneck values.

Returns:

Numpy array of values produced by the bottleneck layer for the image.

"""

label\_lists = image\_lists[label\_name]

sub\_dir = label\_lists['dir']

sub\_dir\_path = os.path.join(bottleneck\_dir, sub\_dir)

ensure\_dir\_exists(sub\_dir\_path)

bottleneck\_path = get\_bottleneck\_path(image\_lists, label\_name, index,

bottleneck\_dir, category)

if not os.path.exists(bottleneck\_path):

create\_bottleneck\_file(bottleneck\_path, image\_lists, label\_name, index,

image\_dir, category, sess, jpeg\_data\_tensor,

bottleneck\_tensor)

with open(bottleneck\_path, 'r') as bottleneck\_file:

bottleneck\_string = bottleneck\_file.read()

did\_hit\_error = False

try:

bottleneck\_values = [float(x) for x in bottleneck\_string.split(',')]

except ValueError:

print('Invalid float found, recreating bottleneck')

did\_hit\_error = True

if did\_hit\_error:

create\_bottleneck\_file(bottleneck\_path, image\_lists, label\_name, index,

image\_dir, category, sess, jpeg\_data\_tensor,

bottleneck\_tensor)

with open(bottleneck\_path, 'r') as bottleneck\_file:

bottleneck\_string = bottleneck\_file.read()

# Allow exceptions to propagate here, since they shouldn't happen after a

# fresh creation

bottleneck\_values = [float(x) for x in bottleneck\_string.split(',')]

return bottleneck\_values

def cache\_bottlenecks(sess, image\_lists, image\_dir, bottleneck\_dir,

jpeg\_data\_tensor, bottleneck\_tensor):

"""Ensures all the training, testing, and validation bottlenecks are cached.

Because we're likely to read the same image multiple times (if there are no

distortions applied during training) it can speed things up a lot if we

calculate the bottleneck layer values once for each image during

preprocessing, and then just read those cached values repeatedly during

training. Here we go through all the images we've found, calculate those

values, and save them off.

Args:

sess: The current active TensorFlow Session.

image\_lists: Dictionary of training images for each label.

image\_dir: Root folder string of the subfolders containing the training

images.

bottleneck\_dir: Folder string holding cached files of bottleneck values.

jpeg\_data\_tensor: Input tensor for jpeg data from file.

bottleneck\_tensor: The penultimate output layer of the graph.

Returns:

Nothing.

"""

how\_many\_bottlenecks = 0

ensure\_dir\_exists(bottleneck\_dir)

for label\_name, label\_lists in image\_lists.items():

for category in ['training', 'testing', 'validation']:

category\_list = label\_lists[category]

for index, unused\_base\_name in enumerate(category\_list):

get\_or\_create\_bottleneck(sess, image\_lists, label\_name, index,

image\_dir, category, bottleneck\_dir,

jpeg\_data\_tensor, bottleneck\_tensor)

how\_many\_bottlenecks += 1

if how\_many\_bottlenecks % 100 == 0:

print(str(how\_many\_bottlenecks) + ' bottleneck files created.')

def get\_random\_cached\_bottlenecks(sess, image\_lists, how\_many, category,

bottleneck\_dir, image\_dir, jpeg\_data\_tensor,

bottleneck\_tensor):

"""Retrieves bottleneck values for cached images.

If no distortions are being applied, this function can retrieve the cached

bottleneck values directly from disk for images. It picks a random set of

images from the specified category.

Args:

sess: Current TensorFlow Session.

image\_lists: Dictionary of training images for each label.

how\_many: If positive, a random sample of this size will be chosen.

If negative, all bottlenecks will be retrieved.

category: Name string of which set to pull from - training, testing, or

validation.

bottleneck\_dir: Folder string holding cached files of bottleneck values.

image\_dir: Root folder string of the subfolders containing the training

images.

jpeg\_data\_tensor: The layer to feed jpeg image data into.

bottleneck\_tensor: The bottleneck output layer of the CNN graph.

Returns:

List of bottleneck arrays, their corresponding ground truths, and the

relevant filenames.

"""

class\_count = len(image\_lists.keys())

bottlenecks = []

ground\_truths = []

filenames = []

if how\_many >= 0:

# Retrieve a random sample of bottlenecks.

for unused\_i in range(how\_many):

label\_index = random.randrange(class\_count)

label\_name = list(image\_lists.keys())[label\_index]

image\_index = random.randrange(MAX\_NUM\_IMAGES\_PER\_CLASS + 1)

image\_name = get\_image\_path(image\_lists, label\_name, image\_index,

image\_dir, category)

bottleneck = get\_or\_create\_bottleneck(sess, image\_lists, label\_name,

image\_index, image\_dir, category,

bottleneck\_dir, jpeg\_data\_tensor,

bottleneck\_tensor)

ground\_truth = np.zeros(class\_count, dtype=np.float32)

ground\_truth[label\_index] = 1.0

bottlenecks.append(bottleneck)

ground\_truths.append(ground\_truth)

filenames.append(image\_name)

else:

# Retrieve all bottlenecks.

for label\_index, label\_name in enumerate(image\_lists.keys()):

for image\_index, image\_name in enumerate(

image\_lists[label\_name][category]):

image\_name = get\_image\_path(image\_lists, label\_name, image\_index,

image\_dir, category)

bottleneck = get\_or\_create\_bottleneck(sess, image\_lists, label\_name,

image\_index, image\_dir, category,

bottleneck\_dir, jpeg\_data\_tensor,

bottleneck\_tensor)

ground\_truth = np.zeros(class\_count, dtype=np.float32)

ground\_truth[label\_index] = 1.0

bottlenecks.append(bottleneck)

ground\_truths.append(ground\_truth)

filenames.append(image\_name)

return bottlenecks, ground\_truths, filenames

def get\_random\_distorted\_bottlenecks(

sess, image\_lists, how\_many, category, image\_dir, input\_jpeg\_tensor,

distorted\_image, resized\_input\_tensor, bottleneck\_tensor):

"""Retrieves bottleneck values for training images, after distortions.

If we're training with distortions like crops, scales, or flips, we have to

recalculate the full model for every image, and so we can't use cached

bottleneck values. Instead we find random images for the requested category,

run them through the distortion graph, and then the full graph to get the

bottleneck results for each.

Args:

sess: Current TensorFlow Session.

image\_lists: Dictionary of training images for each label.

how\_many: The integer number of bottleneck values to return.

category: Name string of which set of images to fetch - training, testing,

or validation.

image\_dir: Root folder string of the subfolders containing the training

images.

input\_jpeg\_tensor: The input layer we feed the image data to.

distorted\_image: The output node of the distortion graph.

resized\_input\_tensor: The input node of the recognition graph.

bottleneck\_tensor: The bottleneck output layer of the CNN graph.

Returns:

List of bottleneck arrays and their corresponding ground truths.

"""

class\_count = len(image\_lists.keys())

bottlenecks = []

ground\_truths = []

for unused\_i in range(how\_many):

label\_index = random.randrange(class\_count)

label\_name = list(image\_lists.keys())[label\_index]

image\_index = random.randrange(MAX\_NUM\_IMAGES\_PER\_CLASS + 1)

image\_path = get\_image\_path(image\_lists, label\_name, image\_index, image\_dir,

category)

if not gfile.Exists(image\_path):

tf.logging.fatal('File does not exist %s', image\_path)

jpeg\_data = gfile.FastGFile(image\_path, 'rb').read()

# Note that we materialize the distorted\_image\_data as a numpy array before

# sending running inference on the image. This involves 2 memory copies and

# might be optimized in other implementations.

distorted\_image\_data = sess.run(distorted\_image,

{input\_jpeg\_tensor: jpeg\_data})

bottleneck = run\_bottleneck\_on\_image(sess, distorted\_image\_data,

resized\_input\_tensor,

bottleneck\_tensor)

ground\_truth = np.zeros(class\_count, dtype=np.float32)

ground\_truth[label\_index] = 1.0

bottlenecks.append(bottleneck)

ground\_truths.append(ground\_truth)

return bottlenecks, ground\_truths

def should\_distort\_images(flip\_left\_right, random\_crop, random\_scale,

random\_brightness):

"""Whether any distortions are enabled, from the input flags.

Args:

flip\_left\_right: Boolean whether to randomly mirror images horizontally.

random\_crop: Integer percentage setting the total margin used around the

crop box.

random\_scale: Integer percentage of how much to vary the scale by.

random\_brightness: Integer range to randomly multiply the pixel values by.

Returns:

Boolean value indicating whether any distortions should be applied.

"""

return (flip\_left\_right or (random\_crop != 0) or (random\_scale != 0) or

(random\_brightness != 0))

def add\_input\_distortions(flip\_left\_right, random\_crop, random\_scale,

random\_brightness):

"""Creates the operations to apply the specified distortions.

During training it can help to improve the results if we run the images

through simple distortions like crops, scales, and flips. These reflect the

kind of variations we expect in the real world, and so can help train the

model to cope with natural data more effectively. Here we take the supplied

parameters and construct a network of operations to apply them to an image.

Cropping

~~~~~~~~

Cropping is done by placing a bounding box at a random position in the full

image. The cropping parameter controls the size of that box relative to the

input image. If it's zero, then the box is the same size as the input and no

cropping is performed. If the value is 50%, then the crop box will be half the

width and height of the input. In a diagram it looks like this:

< width >

+---------------------+

| |

| width - crop% |

| < > |

| +------+ |

| | | |

| | | |

| | | |

| +------+ |

| |

| |

+---------------------+

Scaling

~~~~~~~

Scaling is a lot like cropping, except that the bounding box is always

centered and its size varies randomly within the given range. For example if

the scale percentage is zero, then the bounding box is the same size as the

input and no scaling is applied. If it's 50%, then the bounding box will be in

a random range between half the width and height and full size.

Args:

flip\_left\_right: Boolean whether to randomly mirror images horizontally.

random\_crop: Integer percentage setting the total margin used around the

crop box.

random\_scale: Integer percentage of how much to vary the scale by.

random\_brightness: Integer range to randomly multiply the pixel values by.

graph.

Returns:

The jpeg input layer and the distorted result tensor.

"""

jpeg\_data = tf.placeholder(tf.string, name='DistortJPGInput')

decoded\_image = tf.image.decode\_jpeg(jpeg\_data, channels=MODEL\_INPUT\_DEPTH)

decoded\_image\_as\_float = tf.cast(decoded\_image, dtype=tf.float32)

decoded\_image\_4d = tf.expand\_dims(decoded\_image\_as\_float, 0)

margin\_scale = 1.0 + (random\_crop / 100.0)

resize\_scale = 1.0 + (random\_scale / 100.0)

margin\_scale\_value = tf.constant(margin\_scale)

resize\_scale\_value = tf.random\_uniform(tensor\_shape.scalar(),

minval=1.0,

maxval=resize\_scale)

scale\_value = tf.multiply(margin\_scale\_value, resize\_scale\_value)

precrop\_width = tf.multiply(scale\_value, MODEL\_INPUT\_WIDTH)

precrop\_height = tf.multiply(scale\_value, MODEL\_INPUT\_HEIGHT)

precrop\_shape = tf.stack([precrop\_height, precrop\_width])

precrop\_shape\_as\_int = tf.cast(precrop\_shape, dtype=tf.int32)

precropped\_image = tf.image.resize\_bilinear(decoded\_image\_4d,

precrop\_shape\_as\_int)

precropped\_image\_3d = tf.squeeze(precropped\_image, squeeze\_dims=[0])

cropped\_image = tf.random\_crop(precropped\_image\_3d,

[MODEL\_INPUT\_HEIGHT, MODEL\_INPUT\_WIDTH,

MODEL\_INPUT\_DEPTH])

if flip\_left\_right:

flipped\_image = tf.image.random\_flip\_left\_right(cropped\_image)

else:

flipped\_image = cropped\_image

brightness\_min = 1.0 - (random\_brightness / 100.0)

brightness\_max = 1.0 + (random\_brightness / 100.0)

brightness\_value = tf.random\_uniform(tensor\_shape.scalar(),

minval=brightness\_min,

maxval=brightness\_max)

brightened\_image = tf.multiply(flipped\_image, brightness\_value)

distort\_result = tf.expand\_dims(brightened\_image, 0, name='DistortResult')

return jpeg\_data, distort\_result

def variable\_summaries(var):

"""Attach a lot of summaries to a Tensor (for TensorBoard visualization)."""

with tf.name\_scope('summaries'):

mean = tf.reduce\_mean(var)

tf.summary.scalar('mean', mean)

with tf.name\_scope('stddev'):

stddev = tf.sqrt(tf.reduce\_mean(tf.square(var - mean)))

tf.summary.scalar('stddev', stddev)

tf.summary.scalar('max', tf.reduce\_max(var))

tf.summary.scalar('min', tf.reduce\_min(var))

tf.summary.histogram('histogram', var)

def add\_final\_training\_ops(class\_count, final\_tensor\_name, bottleneck\_tensor):

"""Adds a new softmax and fully-connected layer for training.

We need to retrain the top layer to identify our new classes, so this function

adds the right operations to the graph, along with some variables to hold the

weights, and then sets up all the gradients for the backward pass.

The set up for the softmax and fully-connected layers is based on:

https://tensorflow.org/versions/master/tutorials/mnist/beginners/index.html

Args:

class\_count: Integer of how many categories of things we're trying to

recognize.

final\_tensor\_name: Name string for the new final node that produces results.

bottleneck\_tensor: The output of the main CNN graph.

Returns:

The tensors for the training and cross entropy results, and tensors for the

bottleneck input and ground truth input.

"""

with tf.name\_scope('input'):

bottleneck\_input = tf.placeholder\_with\_default(

bottleneck\_tensor, shape=[None, BOTTLENECK\_TENSOR\_SIZE],

name='BottleneckInputPlaceholder')

ground\_truth\_input = tf.placeholder(tf.float32,

[None, class\_count],

name='GroundTruthInput')

# Organizing the following ops as `final\_training\_ops` so they're easier

# to see in TensorBoard

layer\_name = 'final\_training\_ops'

with tf.name\_scope(layer\_name):

with tf.name\_scope('weights'):

initial\_value = tf.truncated\_normal([BOTTLENECK\_TENSOR\_SIZE, class\_count],

stddev=0.001)

layer\_weights = tf.Variable(initial\_value, name='final\_weights')

variable\_summaries(layer\_weights)

with tf.name\_scope('biases'):

layer\_biases = tf.Variable(tf.zeros([class\_count]), name='final\_biases')

variable\_summaries(layer\_biases)

with tf.name\_scope('Wx\_plus\_b'):

logits = tf.matmul(bottleneck\_input, layer\_weights) + layer\_biases

tf.summary.histogram('pre\_activations', logits)

final\_tensor = tf.nn.softmax(logits, name=final\_tensor\_name)

tf.summary.histogram('activations', final\_tensor)

with tf.name\_scope('cross\_entropy'):

cross\_entropy = tf.nn.softmax\_cross\_entropy\_with\_logits(

labels=ground\_truth\_input, logits=logits)

with tf.name\_scope('total'):

cross\_entropy\_mean = tf.reduce\_mean(cross\_entropy)

tf.summary.scalar('cross\_entropy', cross\_entropy\_mean)

with tf.name\_scope('train'):

optimizer = tf.train.GradientDescentOptimizer(FLAGS.learning\_rate)

train\_step = optimizer.minimize(cross\_entropy\_mean)

return (train\_step, cross\_entropy\_mean, bottleneck\_input, ground\_truth\_input,

final\_tensor)

def add\_evaluation\_step(result\_tensor, ground\_truth\_tensor):

"""Inserts the operations we need to evaluate the accuracy of our results.

Args:

result\_tensor: The new final node that produces results.

ground\_truth\_tensor: The node we feed ground truth data

into.

Returns:

Tuple of (evaluation step, prediction).

"""

with tf.name\_scope('accuracy'):

with tf.name\_scope('correct\_prediction'):

prediction = tf.argmax(result\_tensor, 1)

correct\_prediction = tf.equal(

prediction, tf.argmax(ground\_truth\_tensor, 1))

with tf.name\_scope('accuracy'):

evaluation\_step = tf.reduce\_mean(tf.cast(correct\_prediction, tf.float32))

tf.summary.scalar('accuracy', evaluation\_step)

return evaluation\_step, prediction

def main(\_):

# TensorBoard의 summaries를 write할 directory를 설정한다.

if tf.gfile.Exists(FLAGS.summaries\_dir):

tf.gfile.DeleteRecursively(FLAGS.summaries\_dir)

tf.gfile.MakeDirs(FLAGS.summaries\_dir)

# pre-trained graph를 생성한다.

maybe\_download\_and\_extract()

graph, bottleneck\_tensor, jpeg\_data\_tensor, resized\_image\_tensor = (

create\_inception\_graph())

# 폴더 구조를 살펴보고, 모든 이미지에 대한 lists를 생성한다.

image\_lists = create\_image\_lists(FLAGS.image\_dir, FLAGS.testing\_percentage,

FLAGS.validation\_percentage)

class\_count = len(image\_lists.keys())

if class\_count == 0:

print('No valid folders of images found at ' + FLAGS.image\_dir)

return -1

if class\_count == 1:

print('Only one valid folder of images found at ' + FLAGS.image\_dir +

' - multiple classes are needed for classification.')

return -1

# 커맨드라인 flag에 distortion에 관련된 설정이 있으면 distortion들을 적용한다.

do\_distort\_images = should\_distort\_images(

FLAGS.flip\_left\_right, FLAGS.random\_crop, FLAGS.random\_scale,

FLAGS.random\_brightness)

with tf.Session(graph=graph) as sess:

if do\_distort\_images:

# 우리는 distortion들을 적용할것이다. 따라서 필요한 연산들(operations)을 설정한다.

(distorted\_jpeg\_data\_tensor,

distorted\_image\_tensor) = add\_input\_distortions(

FLAGS.flip\_left\_right, FLAGS.random\_crop,

FLAGS.random\_scale, FLAGS.random\_brightness)

else:

# 우리는 계산된 'bottleneck' 이미지 summaries를 가지고 있다.

# 이를 disk에 캐싱(caching)할 것이다.

cache\_bottlenecks(sess, image\_lists, FLAGS.image\_dir,

FLAGS.bottleneck\_dir, jpeg\_data\_tensor,

bottleneck\_tensor)

# 우리가 학습시킬(training) 새로운 layer를 추가한다.

(train\_step, cross\_entropy, bottleneck\_input, ground\_truth\_input,

final\_tensor) = add\_final\_training\_ops(len(image\_lists.keys()),

FLAGS.final\_tensor\_name,

bottleneck\_tensor)

# 우리의 새로운 layer의 정확도를 평가(evalute)하기 위한 새로운 operation들을 생성한다.

evaluation\_step, prediction = add\_evaluation\_step(

final\_tensor, ground\_truth\_input)

# 모든 summaries를 합치고(merge) summaries\_dir에 쓴다.(write)

merged = tf.summary.merge\_all()

train\_writer = tf.summary.FileWriter(FLAGS.summaries\_dir + '/train',

sess.graph)

validation\_writer = tf.summary.FileWriter(

FLAGS.summaries\_dir + '/validation')

# 우리의 모든 가중치들(weights)과 그들의 초기값들을 설정한다.

init = tf.global\_variables\_initializer()

sess.run(init)

# 커맨드 라인에서 지정한 횟수만큼 학습을 진행한다.

for i in range(FLAGS.how\_many\_training\_steps):

# bottleneck 값들의 batch를 얻는다. 이는 매번 distortion을 적용하고 계산하거나,

# disk에 저장된 chache로부터 얻을 수 있다.

if do\_distort\_images:

(train\_bottlenecks,

train\_ground\_truth) = get\_random\_distorted\_bottlenecks(

sess, image\_lists, FLAGS.train\_batch\_size, 'training',

FLAGS.image\_dir, distorted\_jpeg\_data\_tensor,

distorted\_image\_tensor, resized\_image\_tensor, bottleneck\_tensor)

else:

(train\_bottlenecks,

train\_ground\_truth, \_) = get\_random\_cached\_bottlenecks(

sess, image\_lists, FLAGS.train\_batch\_size, 'training',

FLAGS.bottleneck\_dir, FLAGS.image\_dir, jpeg\_data\_tensor,

bottleneck\_tensor)

# grpah에 bottleneck과 ground truth를 feed하고, training step을 진행한다.

# TensorBoard를 위한 'merged' op을 이용해서 training summaries을 capture한다.

train\_summary, \_ = sess.run(

[merged, train\_step],

feed\_dict={bottleneck\_input: train\_bottlenecks,

ground\_truth\_input: train\_ground\_truth})

train\_writer.add\_summary(train\_summary, i)

# 일정 step마다 graph의 training이 얼마나 잘 되고 있는지 출력한다.

is\_last\_step = (i + 1 == FLAGS.how\_many\_training\_steps)

if (i % FLAGS.eval\_step\_interval) == 0 or is\_last\_step:

train\_accuracy, cross\_entropy\_value = sess.run(

[evaluation\_step, cross\_entropy],

feed\_dict={bottleneck\_input: train\_bottlenecks,

ground\_truth\_input: train\_ground\_truth})

print('%s: Step %d: Train accuracy = %.1f%%' % (datetime.now(), i,

train\_accuracy \* 100))

print('%s: Step %d: Cross entropy = %f' % (datetime.now(), i,

cross\_entropy\_value))

validation\_bottlenecks, validation\_ground\_truth, \_ = (

get\_random\_cached\_bottlenecks(

sess, image\_lists, FLAGS.validation\_batch\_size, 'validation',

FLAGS.bottleneck\_dir, FLAGS.image\_dir, jpeg\_data\_tensor,

bottleneck\_tensor))

# validation step을 진행한다.

# TensorBoard를 위한 'merged' op을 이용해서 training summaries을 capture한다.

validation\_summary, validation\_accuracy = sess.run(

[merged, evaluation\_step],

feed\_dict={bottleneck\_input: validation\_bottlenecks,

ground\_truth\_input: validation\_ground\_truth})

validation\_writer.add\_summary(validation\_summary, i)

print('%s: Step %d: Validation accuracy = %.1f%% (N=%d)' %

(datetime.now(), i, validation\_accuracy \* 100,

len(validation\_bottlenecks)))

# 트레이닝 과정이 모두 끝났다.

# 따라서 이전에 보지 못했던 이미지를 통해 마지막 test 평가(evalution)을 진행한다.

test\_bottlenecks, test\_ground\_truth, test\_filenames = (

get\_random\_cached\_bottlenecks(sess, image\_lists, FLAGS.test\_batch\_size,

'testing', FLAGS.bottleneck\_dir,

FLAGS.image\_dir, jpeg\_data\_tensor,

bottleneck\_tensor))

test\_accuracy, predictions = sess.run(

[evaluation\_step, prediction],

feed\_dict={bottleneck\_input: test\_bottlenecks,

ground\_truth\_input: test\_ground\_truth})

print('Final test accuracy = %.1f%% (N=%d)' % (

test\_accuracy \* 100, len(test\_bottlenecks)))

if FLAGS.print\_misclassified\_test\_images:

print('=== MISCLASSIFIED TEST IMAGES ===')

for i, test\_filename in enumerate(test\_filenames):

if predictions[i] != test\_ground\_truth[i].argmax():

print('%70s %s' % (test\_filename,

list(image\_lists.keys())[predictions[i]]))

# 학습된 graph와 weights들을 포함한 labels를 쓴다.(write)

output\_graph\_def = graph\_util.convert\_variables\_to\_constants(

sess, graph.as\_graph\_def(), [FLAGS.final\_tensor\_name])

with gfile.FastGFile(FLAGS.output\_graph, 'wb') as f:

f.write(output\_graph\_def.SerializeToString())

with gfile.FastGFile(FLAGS.output\_labels, 'w') as f:

f.write('\n'.join(image\_lists.keys()) + '\n')

if \_\_name\_\_ == '\_\_main\_\_':

parser = argparse.ArgumentParser()

parser.add\_argument(

'--image\_dir',

type=str,

default='../DB',

help='Path to folders of labeled images.'

)

parser.add\_argument(

'--output\_graph',

type=str,

default='./learned\_model/output\_graph.pb',

help='Where to save the trained graph.'

)

parser.add\_argument(

'--output\_labels',

type=str,

default='./learned\_model/output\_labels.txt',

help='Where to save the trained graph\'s labels.'

)

parser.add\_argument(

'--summaries\_dir',

type=str,

default='./learned\_model/retrain\_logs',

help='Where to save summary logs for TensorBoard.'

)

parser.add\_argument(

'--how\_many\_training\_steps',

type=int,

default=1000,

help='How many training steps to run before ending.'

)

parser.add\_argument(

'--learning\_rate',

type=float,

default=0.01,

help='How large a learning rate to use when training.'

)

parser.add\_argument(

'--testing\_percentage',

type=int,

default=10,

help='What percentage of images to use as a test set.'

)

parser.add\_argument(

'--validation\_percentage',

type=int,

default=10,

help='What percentage of images to use as a validation set.'

)

parser.add\_argument(

'--eval\_step\_interval',

type=int,

default=10,

help='How often to evaluate the training results.'

)

parser.add\_argument(

'--train\_batch\_size',

type=int,

default=100,

help='How many images to train on at a time.'

)

parser.add\_argument(

'--test\_batch\_size',

type=int,

default=-1,

help="""\

How many images to test on. This test set is only used once, to evaluate

the final accuracy of the model after training completes.

A value of -1 causes the entire test set to be used, which leads to more

stable results across runs.\

"""

)

parser.add\_argument(

'--validation\_batch\_size',

type=int,

default=100,

help="""\

How many images to use in an evaluation batch. This validation set is

used much more often than the test set, and is an early indicator of how

accurate the model is during training.

A value of -1 causes the entire validation set to be used, which leads to

more stable results across training iterations, but may be slower on large

training sets.\

"""

)

parser.add\_argument(

'--print\_misclassified\_test\_images',

default=False,

help="""\

Whether to print out a list of all misclassified test images.\

""",

action='store\_true'

)

parser.add\_argument(

'--model\_dir',

type=str,

default='/tmp/imagenet',

help="""\

Path to classify\_image\_graph\_def.pb,

imagenet\_synset\_to\_human\_label\_map.txt, and

imagenet\_2012\_challenge\_label\_map\_proto.pbtxt.\

"""

)

parser.add\_argument(

'--bottleneck\_dir',

type=str,

default='/tmp/bottleneck',

help='Path to cache bottleneck layer values as files.'

)

parser.add\_argument(

'--final\_tensor\_name',

type=str,

default='final\_result',

help="""\

The name of the output classification layer in the retrained graph.\

"""

)

parser.add\_argument(

'--flip\_left\_right',

default=False,

help="""\

Whether to randomly flip half of the training images horizontally.\

""",

action='store\_true'

)

parser.add\_argument(

'--random\_crop',

type=int,

default=0,

help="""\

A percentage determining how much of a margin to randomly crop off the

training images.\

"""

)

parser.add\_argument(

'--random\_scale',

type=int,

default=0,

help="""\

A percentage determining how much to randomly scale up the size of the

training images by.\

"""

)

parser.add\_argument(

'--random\_brightness',

type=int,

default=0,

help="""\

A percentage determining how much to randomly multiply the training image

input pixels up or down by.\

"""

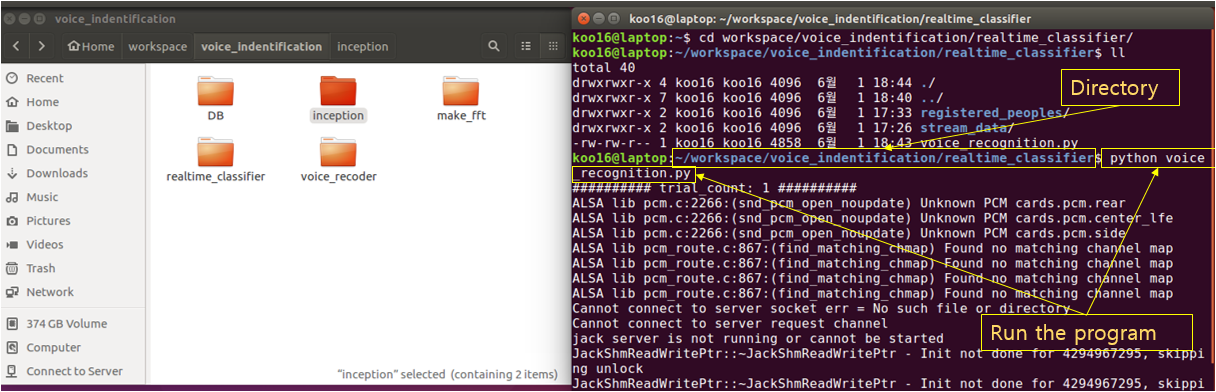
)

FLAGS, unparsed = parser.parse\_known\_args()

tf.app.run(main=main, argv=[sys.argv[0]] + unparsed)

**5.5 Step 5** **“How to perform real-time demonstration?”**

Run “voice\_recognition.py” program in Linux terminal to perform real-time demonstration using trained Inception V3 model.



# voice\_recognition.py

#! /usr/bin/env python

# -\*- coding: utf-8 -\*-

**import** numpy **as** np

**import** tensorflow **as** tf

**import** matplotlib**.**pyplot **as** plt

**from** scipy**.**io **import** wavfile

**import** pyaudio

**import** wave

**import** cv2

**class** **Recorder(**object**):**

**def** \_\_init\_\_**(**self**,** channels**=**1**,** rate**=**44100**,** frames\_per\_buffer**=**1024**):**

self**.**channels **=** channels

self**.**rate **=** rate

self**.**frames\_per\_buffer **=** frames\_per\_buffer

**def** open**(**self**,** fname**,** mode**=**'wb'**):**

**return** RecordingFile**(**fname**,** mode**,** self**.**channels**,** self**.**rate**,**

self**.**frames\_per\_buffer**)**

**class** **RecordingFile(**object**):**

**def** \_\_init\_\_**(**self**,** fname**,** mode**,** channels**,**

rate**,** frames\_per\_buffer**):**

self**.**fname **=** fname

self**.**mode **=** mode

self**.**channels **=** channels

self**.**rate **=** rate

self**.**frames\_per\_buffer **=** frames\_per\_buffer

self**.**\_pa **=** pyaudio**.**PyAudio**()**

self**.**wavefile **=** self**.**\_prepare\_file**(**self**.**fname**,** self**.**mode**)**

self**.**\_stream **=** **None**

**def** \_\_enter\_\_**(**self**):**

**return** self

**def** \_\_exit\_\_**(**self**,** exception**,** value**,** traceback**):**

self**.**close**()**

def record(self, duration):

self.\_stream = self.\_pa.open(format=pyaudio.paInt16,

channels=self.channels,

rate=self.rate,

input=True,

frames\_per\_buffer=self.frames\_per\_buffer)

for \_ in range(int(self.rate / self.frames\_per\_buffer \* duration)):

audio = self.\_stream.read(self.frames\_per\_buffer)

self.wavefile.writeframes(audio)

return None

def stop\_recording(self):

self.\_stream.stop\_stream()

return self

def get\_callback(self):

def callback(in\_data, frame\_count, time\_info, status):

self.wavefile.writeframes(in\_data)

return in\_data, pyaudio.paContinue

return callback

def close(self):

self.\_stream.close()

self.\_pa.terminate()

self.wavefile.close()

def \_prepare\_file(self, fname, mode='wb'):

wavefile = wave.open(fname, mode)

wavefile.setnchannels(self.channels)

wavefile.setsampwidth(self.\_pa.get\_sample\_size(pyaudio.paInt16))

wavefile.setframerate(self.rate)

return wavefile

def convert\_spectrogram():

# Make spectrogram using stream voice file

wav\_file\_path = './stream\_data/record\_file.wav'

postprocessing\_image\_path = './stream\_data/spectrogram\_image.jpg'

sample\_frequency, signalData = wavfile.read(wav\_file\_path)

plt.specgram(signalData[:, 0], Fs=sample\_frequency)

plt.axis([0, 3, 0, 22000])

plt.savefig(postprocessing\_image\_path)

def create\_graph():

# Create tensorflow graph using pre-saved pb file

with tf.gfile.FastGFile(model\_path, 'rb') as f:

graph\_def = tf.GraphDef()

graph\_def.ParseFromString(f.read())

\_ = tf.import\_graph\_def(graph\_def, name='')

def classifier():

# Load spectrogram

image\_data = tf.gfile.FastGFile(image\_path, 'rb').read()

# Create tensorflow graph using pre-saved pb file

create\_graph()

with tf.Session() as sess:

# Using retrained Inception v3, classify the unknown stream voice

softmax\_tensor = sess.graph.get\_tensor\_by\_name('final\_result:0')

predictions = sess.run(softmax\_tensor,

{'DecodeJpeg/contents:0': image\_data})

predictions = np.squeeze(predictions)

top\_k = predictions.argsort()[-5:][::-1] # Get top 5 prediction

f = open(labels\_path, 'rb')

lines = f.readlines()

labels = [str(w).replace("\n", "") for w in lines]

for node\_id in top\_k:

human\_string = labels[node\_id]

score = predictions[node\_id]

identified\_people.append(human\_string)

print('%s (score = %.5f)' % (human\_string, score))

if \_\_name\_\_ == '\_\_main\_\_':

model\_path = '../inception/learned\_model/output\_graph.pb'

labels\_path = '../inception/learned\_model/output\_labels.txt'

image\_path = './stream\_data/spectrogram\_image.jpg'

identified\_people = []

for trial\_number in range(1, 101, 1):

print ('########## trial\_count: {} ##########'.format(trial\_number))

# Record stream voice and save as wav file

rec = Recorder(channels=2)

with rec.open('./stream\_data/record\_file.wav', 'wb') as recfile:

recfile.record(duration=3.0)

# Convert stream voice to spectrogram

convert\_spectrogram()

# Do classification on stream spectrogram

classifier()

# Make UI for classification result

original = cv2.imread('./registered\_peoples/' + identified\_people[0] + '.png', cv2.IMREAD\_COLOR)

cv2.imshow('Owner of voice', original)

cv2.waitKey(0)

cv2.destroyAllWindows()

identified\_people = []

**References**

1. Why Deep Learning is so popular and in demand, <https://medium.com/swlh/ill-tell-you-why-deep-learning-is-so-popular-and-in-demand-5aca72628780>
2. Deep learning, <https://en.wikipedia.org/wiki/Deep_learning>
3. A History of Deep Learning, <https://www.import.io/post/history-of-deep-learning/>
4. ImageNet, <https://en.wikipedia.org/wiki/ImageNet>
5. Iris Recognition with Off-the-Shelf CNN Features: A Deep Learning Perspective, <https://ieeexplore.ieee.org/document/8219390/>
6. Top 9 Machine Learning Applications in Real World, <https://data-flair.training/blogs/machine-learning-applications/>
7. What is a convolutional neural network? <https://www.quora.com/What-is-a-convolutional-neural-network>
8. Simple Image Classification using Convolutional Neural Network - Deep Learning in python, <https://becominghuman.ai/building-an-image-classifier-using-deep-learning-in-python-totally-from-a-beginners-perspective-be8dbaf22dd8>
9. NIRFaceNet: A Convolutional Neural Network for Near-Infrared Face Identification, <http://www.mdpi.com/2078-2489/7/4/61/htm>
10. ImageNet: VGGNet, ResNet, Inception, and Xception with Keras, <https://www.pyimagesearch.com/2017/03/20/imagenet-vggnet-resnet-inception-xception-keras/>
11. Rethinking the Inception Architecture for Computer Vision, <https://arxiv.org/abs/1512.00567>
12. Advanced Guide to Inception v3 on Cloud TPU, <https://cloud.google.com/tpu/docs/inception-v3-advanced>
13. <https://gist.github.com/sloria/5693955>
14. <https://stackoverflow.com/questions/23377665/python-scipy-fft-wav-files>
15. <http://solarisailab.com/archives/1422>