MSDS 7346 – Cloud Computing

Facial Recognition and Identification of Refugees for Event Registration

Term Project

Brian Lee, Mooyoung Lee, Vivek Bejugama

8-11-2018

Introduction

In 2016, 85,000 refugees were admitted into the US and then 54,000 refugees admitted in 2017 (Solis 2018). In the past, the US had been a leader in providing global support for refugees. That number has been reduced due to political influences and reallocation of national resources; however, the issue of the support provided to refugees are still as much of a challenge as ever. Housing, education, food, etc. all become a issue of logistics when provided support to refugees. The states of Texas, California, and New York account for a nearly a quarter for resettlement for refugees entering the US in 2016 (Krogstad and Radford 2017). As a result, there is a growing need for community services to support the families and the children of the refugees.

Dallas is station to "Love Is Ministry," a non-profit organization with a commitment to support refugees integrate and resettle into Dallas communities ("Love Is Ministrity" 2018). Refugees face many problems integrating in their resettlement areas. Some of these challenges these changes can include language, cultural barriers, employment, education, discrimination, and racism (Hattab 2016). Love Is Ministry helps alleviate some of these hardships by providing after school learning, adult learning, citizenship classes, and mentorships. In addition to the previous services mentioned, a summer soccer league is also part of the program that give the children of refugees and opportunity to gather and engage among the rest of community. We aim to provide support to the summer soccer league after identifying major operational issues that cause an incredible amount effort from event organizers.

The Problem

Potentially hundreds of children show up to the Summer Soccer League every Saturday. Of course, because this is a community event, no children are turned away. Naturally, for the sake of organization, kids are provided shirts with colors corresponding to the teams they have registered for the event. The current system is setup with identifying information for the children that includes first name, last name, team, picture, and permission for contacting emergency services in the event of an emergency. The major challenge of this the registration process prior to the beginning of the games. Registration can become so chaotic that it can potentially take hours before the first game is even begun on the field.

Compounding in addition to typical operational issues, all registrants of the event are children, some with little to no ability of speaking English as English is not their first languages. Because of these challenges, duplicate entries are created in the existing database to ensure that the child has been registered; however, the names are not always spelled the same and other information is not consistent. There is no unique identifier that the kids can provide, nor is the data structured to support any ID for the children. The true “unique” identifier is the image of the child.

We will be creating a cloud architecture to process the images of the children so we can revamp the registration process to be able to identify children that have already been registered and/or have not registered and will require registration. The ability for the children to be easily identified will greatly enhance the speed of the registration process.

Method

We create a cloud architecture that allows for data storage and processing of the images that have been gathered. The challenges faced, as expected in a real-world environment, are to gain the permissions to access and process the data that was requested. The next sections below describe the findings and the methods that were explored to develop a working model as a proof of concept.

Cloud Architecture - Current

Figure 1 below is a high-level diagram depicting the current infrastructure. The method that the volunteers were utilizing was to manually take a photo of the event participant, then to attach to a form, which also had fields for participant information including name, number, team, address, permission form for contacting emergency services, etc. The required data works well; however, due to the operational challenges surrounding the inability for many of the participants ability to speak English well, resulting in misspelling of names, duplicating data and other mistakes due to simply attempting to brute force through the registration process. Adding to level of difficulty, children that have not been registered, must be registered on the spot. This can occur with individual children joining the teams or entire teams that require registration.

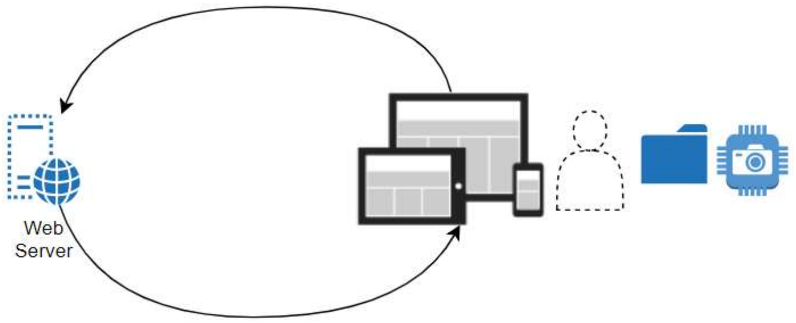


Figure 1: Current registration system/infrastructure

Cloud Architecture - Proposed

The proposed infrastructure can be seen below in Figure 2. The develop of our proof on concept does not include the development of our delivery system, which is a critical component of the overall project. For the scope of our project we look at the utilization of cloud solution on the back end to determine how this may appear in a production environment.

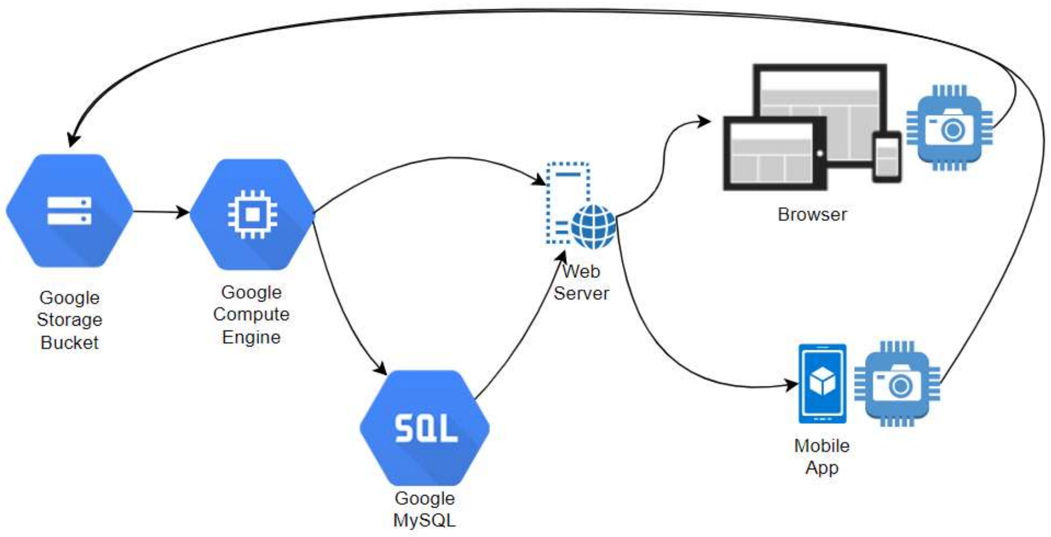


Figure 2: Proposed registration system/infrastructure with image recognition

Previously, the image of the event participants would simply be stored on the webserver, so the event volunteer can visually verify the participant picture matches with their name. This method requires visual inspection by the event volunteer. With the proposed infrastructure, the even participant will stand in front of a camera that would take their image which is then triggers sequences that will send that image to a Google cloud storage bucket along with any form data. This data will then be passed through our Google Compute Engine with our machine learning algorithm to update the model and output to a database and/or to the webserver.

Algorithm

Local Binary Patterns Histograms (LBPH)

LBPH is a simple yet powerful algorithm to conver image into a grayscale vector. It takes frequency of brightness from a grid section of image and add all frequency values to make the final vector.

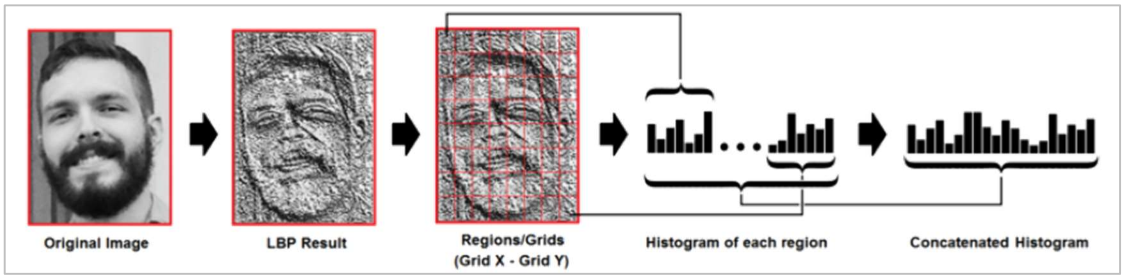


Figure 3. LBPH Image to Histogram Vector Transition

[source: https://towardsdatascience.com/face-recognition-how-lbph-works-90ec258c3d6b]

Convolutional Neural Network (CNN) models

CNN model is the most common type of deep learning model for image classification task. It include convolution, pooling, and fully-connected layer [Zhang]. Convolutional layer apply filters to transform images, and the pooling layer reduce the dimensions of data. The final fully-connected layer provide features to classify image for the softmax layer as shown below image. Normally, deep CNN model include multiple convolutional layers and pooling layers. CNN model shows good performance on other application domains recently such as language processing and speech recognition.

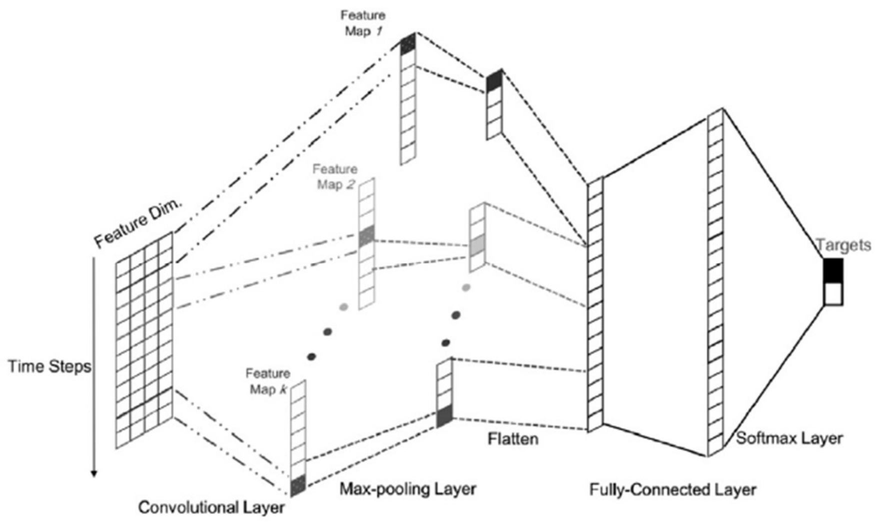


Figure 4: Convolutional Neural Network

Transfer Learning

Transfer learning in machine learning is useful when we have not sufficient amount of data to train for a domain that we are interested but when we have sufficient amount of data to train for another domain [Pan]. In that case, knowledge transfer helps to improve a classification task tremendously without a large amount of data in the domain of interest. Transfer learning have been used only for limited areas such as image classification, text classification, and sensor-network-based localization. According to Pan's survey paper, the transfer learning can be classified as three different types by the availability of the labels in the source and the target domains as shown below. Our face identification task belongs to the inductive transfer learning since the original VGG16 model is trained using labeled ImageNet dataset and the kids at the soccer league have their names. The main knowledge that we are transferring from the VGG16 architecture is the feature representation skills. Since the VGG16 model is trained to have a nice filter sets to find edges, lines, and shapes of images, we can just reuse their knowledge of extracting features. In transfer modeling process, the pre-trained model will be loaded and the most of weights will be frozen so that the feature extraction ability stays same.

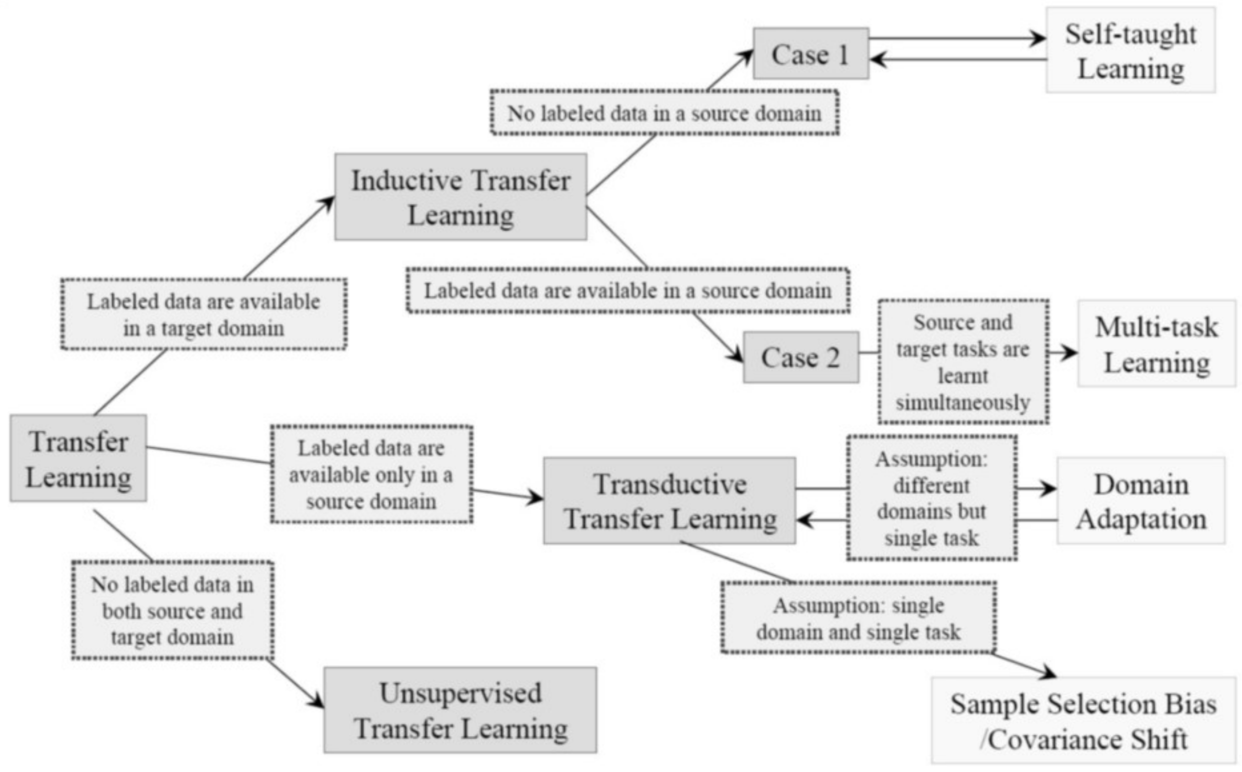

Case 1
Self-taught Learning
No labeled data in a source domain
Inductive Transfer Learning
Labeled data are available in a target domain
Labeled data are available in a source domain
Case 2
Source and target tasks are learnt simultaneously
Multi-task Learning
Transfer Learning
Labeled data are available only in a source domain
Transductive Transfer Learning
Assumption: different domains but single task
Domain Adaptation
No labeled data in both source and target domain
Assumption: single domain and single task
Unsupervised Transfer Learning
Sample Selection Bias /Covariance Shift

Figure 5. An overview of different settings of transfer [PAN]

VGG16

VGG16 is a neural net architecture designed by Visual Geometry Group of Oxford, so also called as OxfordNet which won the ImageNet competition in 2014. VGG16 has 16 layers of neural network. This is a pre-trained model which can be used to predict new images by changed parameter depending on the desired accuracy and processing time.

In the below image, the upsampled outputs of a particular layer are concatenated with the outputs of the previous layer to improve the accuracy of the output.

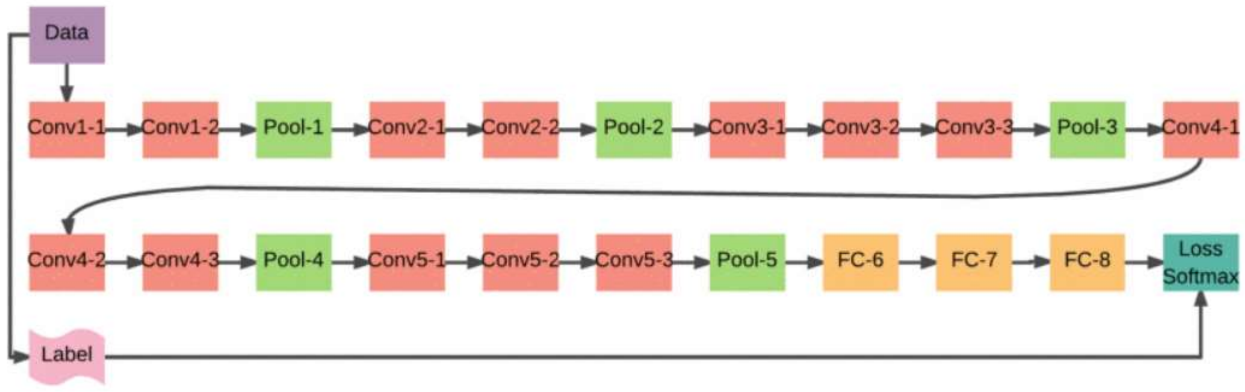
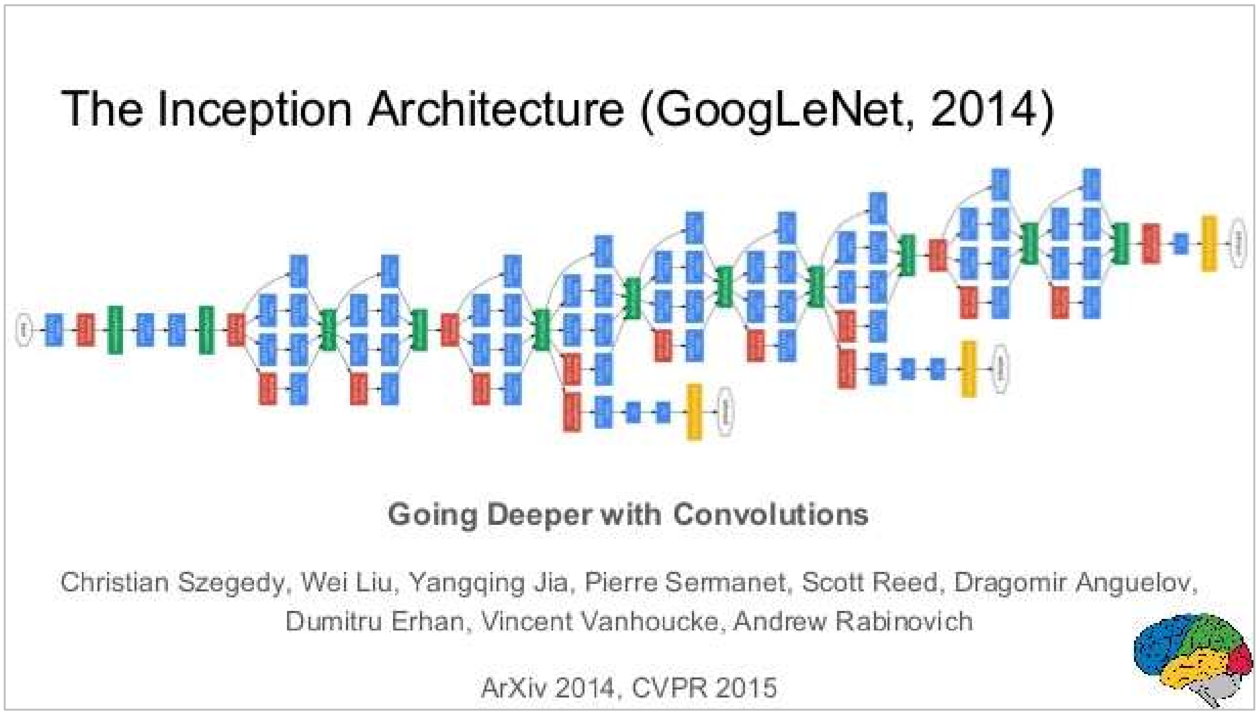


Figure 6. VGG16 Architecture [Qassim]

InceptionV3

The idea of Inception was introduced by GoogLeNet in 2014 by winning the hardest ImageNet challenge for object classification task known as the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) [Alom]. The GoogLeNet architecture as shown below image include inception layer and made the computation easy and also improved performance comparing to the traditional CNN architectures. GoogLeNet uses only 4Mb for network parameters and memory, whereas, the conventional CNN (e.g. AlexNet) requires 60Mb of memory for parameters. In 2015, the Inception model is proposed by Szegedy based.



The Inception Architecture (GoogLeNet, 2014)

Going Deeper with Convolutions

Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, Andrew Rabinovich

ArXiv 2014, CVPR 2015

Figure 7. Inception Architecture [Alom]

Analysis and Result

Datasets

The main task of this project is identifying people by using their frontal face images. We were able to collect total 55 frontal face images from our classmates. To train models and validate model performances, the data need to be divided into train and validation datasets. The train and validation datasets have split with 3:2 ratio as shown on Figure 8.

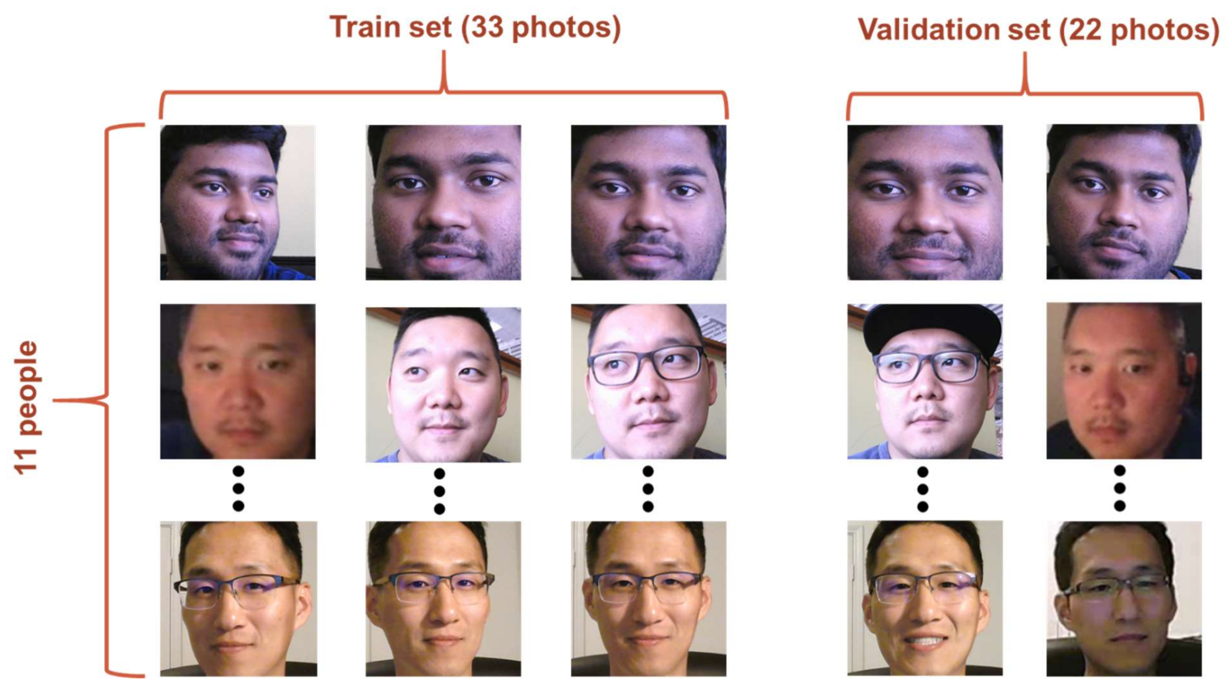
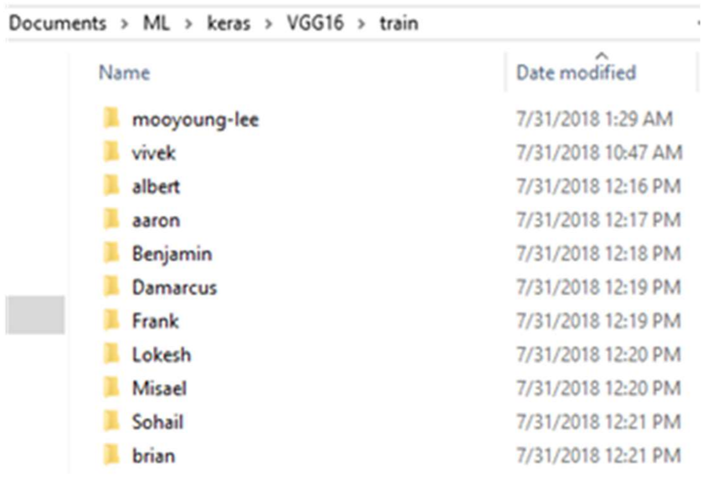


Figure 8. Train and validation datasets

In practice the images of people will be taken using a camera attached to a laptop computer to train a model. Each person will take multiple photos, and the images will be saved under a folder with the corresponding person's name as shown on Figure 9.



Documents > ML > keras > VGG16 > train

Name	Date modified
mooyoung-lee	7/31/2018 1:29 AM
vivek	7/31/2018 10:47 AM
albert	7/31/2018 12:16 PM
aaron	7/31/2018 12:17 PM
Benjamin	7/31/2018 12:18 PM
Damarcus	7/31/2018 12:19 PM
Frank	7/31/2018 12:19 PM
Lokesh	7/31/2018 12:20 PM
Misael	7/31/2018 12:20 PM
Sohail	7/31/2018 12:21 PM
brian	7/31/2018 12:21 PM

Figure 9. Structure of train data set

Python script is written to move image files from camera folder into a train data folder with creating a new folder with a name typed as shown below. This code need to be present to save time and label images during the process of taking photos of kids.

Code 1. Labeling and organizing images

```
name = 'brian'

camera_dir = r'C:\Users\ML\Pictures\Camera Roll'
data_dir = r'C:\Users\ML\Documents\faceID\image'
name_dir = os.path.join(data_dir, name)
# create a target directory for each person
if not os.path.exists(name_dir):
    os.makedirs(name_dir)

for root, dirs, files in os.walk(camera_dir):
    for file in files:
        if file.endswith('jpg') or file.endswith('png'):
            path_src = os.path.join(root, file)
            path_target = os.path.join(name_dir, file)
            print(path_src)
            os.rename(path_src, path_target)
```

Modeling procedure

The environment for the machine learning modeling was Python 3.6 on Windows operating system. For CNN models Keras-gpu 2.2.0 library is used to use the TensorFlow library easily.

A. LBPH model using OpenCV 3.3.1

The LBPH algorithm convert an image into a gray scale histogram that is a 256-long vector. The first step of modeling with this algorithm was looping through all directories in the train data folder and append all images into a matrix. To make image classification task easier, only the frontal face part is selected and used for modeling. Selecting the face area out of a whole image was achieved by Haar feature-based cascade classifiers [source: https://docs.opencv.org/3.4.1/d7/d8b/tutorial_py_face_detection.html]. Then the matrix with arrays containing only face area along with their labels were used to train LBPH model.

Code 2. Training LBPH Model

```
# image file directory
image_dir = r'C:\Users\ML\Documents\faceID\image'

# face classifier
face_cascade = cv2.CascadeClassifier('cascades/data/haarcascade_frontalface_alt2.xml')
recognizer = cv2.face.LBPHFaceRecognizer_create()
# recognizer = cv2.face.createLBPHFaceRecognizer()

current_id = 0
label_ids = {}
x_train = [] # list to hold all subject faces
y_labels = [] # list to hold labels for all subjects

for root, dirs, files in os.walk(image_dir):
    for file in files:
        if file.endswith('jpg') or file.endswith('png'):
            path = os.path.join(root, file)
            label = os.path.basename(os.path.dirname(path)).replace(' ','-').lower()

            # ID nubmer and label dictionary
            if not label in label_ids:
                label_ids[label] = current_id
                current_id += 1

            id_ = label_ids[label]
            pil_image = Image.open(path).convert('L')  # open image and convert to grayscale
            image_array =  np.array(pil_image, 'uint8')

            faces = face_cascade.detectMultiScale(image_array, scaleFactor = 1.5, minNeighbors = 5)

            for (x,y,w,h) in faces:
                roi = image_array[y:y+h, x:x+w]
                x_train.append(roi)
                y_labels.append(id_)

with open('labels_3.pickle', 'wb') as f:
    pickle.dump(label_ids, f)

recognizer.train(x_train, np.array(y_labels))
recognizer.write('trainner_3.yml')
print('Model Completed')
```

After the LBPH model is trained, the model is tested with validation dataset. Testing procedure involved following items in order. First, saved 'yml' file will be loaded into LBPH model object. Seconds, a face part from a test image will be selected and converted into a LBPH vector format. Third, id and confidence values will be calculated using a distance calculation between a train matrix and a test vector.

Code 3. Prediction using LBPH model

```
# load trained model
face_cascade = cv2.CascadeClassifier('cascades/data/haarcascade_frontalface_alt2.xml')
recognizer = cv2.face.LBPHFaceRecognizer_create()
recognizer.read('trainner.yml')

# labeling name
labels = {}
with open('labels.pickle', 'rb') as f:
    old_labels = pickle.load(f)
    labels = {v:k for k, v in old_labels.items()}


def image_recognizer(filename):
    # image read
    frame = cv2.imread(filename, flags = cv2.IMREAD_COLOR)

    # Display grayscale frames
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    faces = face_cascade.detectMultiScale(gray, scaleFactor = 1.5, minNeighbors=5)
    for (x, y, w, h) in faces:

        # region of interest
        roi_gray = gray[y:y+h, x:x+w]
        roi_color = frame[y:y+h, x:x+w]

        # predict
        id_, conf = recognizer.predict(roi_gray)

    try:
        return [labels[id_], round(conf,2)];  # return id and confidence value 0~100-ish
    except:
        return ['NaN', 'NaN']


# Evaluation w/ all validation images

image_dir = r'C:\Users\ML\Documents\faceID\validation'

TP = 0
num_prediction = 0
for root, dirs, files in os.walk(image_dir):
    for file in files:
        if file.endswith('jpg') or file.endswith('PNG'):
            path = os.path.join(root, file)
            label = os.path.basename(os.path.dirname(path)).replace(' ','-').lower()
            print(label, path)
            prediction = image_recognizer(path)
            print(prediction)
            if label == prediction[0]:
                TP += 1
            num_prediction +=1
```

B. CNN model using a Transfer Learning

The procedure of running a transfer learning consist of following steps:

- Load a base model architecture and weights
- Freeze layers from updating weights except last few layers
- Add more customized layers
- Compile the model
- Train the model
- Use the model for prediction/classification

Loading the pre-trained models are easy with Keras library since Keras include various CNN architectures and their weights. One can simply change architecture name by importing different models as shown below. The input data shape is normally 224x224 pixel image with RGB colors.

Code 4. Load Pre-Trained Model

```
# Loading VGG16 architecture
from keras.applications import VGG16
base_model = VGG16(weights='imagenet', include_top=False, input_shape=(224, 224, 3))

# Loading ResNet50
from keras.applications.resnet50 import ResNet50
base_model = ResNet50(weights='imagenet', include_top=False, input_shape=(224, 224, 3))

# Loading InceptionV3
from keras.applications.inception_v3 import InceptionV3
base_model = InceptionV3(weights='imagenet', include_top=False, input_shape=(224, 224, 3))
```

The 'include_top' option allows to include the original fully-connected layer so that the model will work as a default model to classify 1000 images for ImageNet competition. For transfer learning the top layers should not be included. Among the included layers, most of the weights need to be freeze because it has more chance to ruin the feature extraction capability by updating weights. We froze all layers except the last four layers as shown below.

Code 5. Freeze Layers from Pre-Trained Model

```
# Freeze the layers except the last 4 layers
for layer in vgg_conv.layers[:-4]:
   layer.trainable = False

# Check the trainable status of the individual layers
for layer in vgg_conv.layers:
   print(layer, layer.trainable)

<keras.engine.input_layer.InputLayer object at 0x000001C8151DB860> False
<keras.layers.convolutional.Conv2D object at 0x000001C81BB01E80> False
<keras.layers.convolutional.Conv2D object at 0x000001C81BB01F98> False
<keras.layers.pooling.MaxPooling2D object at 0x000001C81BB01FD0> False
<keras.layers.convolutional.Conv2D object at 0x000001C81BB3A470> False
<keras.layers.convolutional.Conv2D object at 0x000001C81BB3AC88> False
<keras.layers.pooling.MaxPooling2D object at 0x000001C81BB75E10> False
<keras.layers.convolutional.Conv2D object at 0x000001C81BB64C18> False
<keras.layers.convolutional.Conv2D object at 0x000001C81BB8AA20> False
<keras.layers.convolutional.Conv2D object at 0x000001C81BBA1B70> False
<keras.layers.pooling.MaxPooling2D object at 0x000001C81BBB4160> False
<keras.layers.convolutional.Conv2D object at 0x000001C81BBC6F98> False
<keras.layers.convolutional.Conv2D object at 0x000001C81BC05780> False
<keras.layers.convolutional.Conv2D object at 0x000001C81BBEE438> False
<keras.layers.pooling.MaxPooling2D object at 0x000001C81BC19160> False
<keras.layers.convolutional.Conv2D object at 0x000001C81BC29EB8> True
<keras.layers.convolutional.Conv2D object at 0x000001C81BC67EF0> True
<keras.layers.convolutional.Conv2D object at 0x000001C81BC55F28> True
<keras.layers.pooling.MaxPooling2D object at 0x000001C81BC74CF8> True
```

Custom model was generated by adding a flatten layer to have a feature vector output from the pre-trained model. The flattened vector output is sent to a dense layer with 1,024-neuron and filtered with an activation function 'relu'. A dropout layer is added to deactivate some neurons and prevent overfitting. And final dense layer is added to classify into a target classification number with 'softmax' activation as shown below.

Code 6. Custom Layers

```
targetClassNumber = 11

# Create the model
model = models.Sequential()

# Add the vgg convolutional base model
model.add(vgg_conv)

# Add new layers
model.add(layers.Flatten())
model.add(layers.Dense(1024, activation='relu'))
model.add(layers.Dropout(0.5))
model.add(layers.Dense(targetClassNumber, activation='softmax'))

# Show a summary of the model. Check the number of trainable parameters
model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
vgg16 (Model)                (None, 7, 7, 512)         14714688  
_________________________________________________________________
flatten_1 (Flatten)          (None, 25088)             0         
_________________________________________________________________
dense_1 (Dense)              (None, 1024)              25691136  
_________________________________________________________________
dropout_1 (Dropout)          (None, 1024)              0         
_________________________________________________________________
dense_2 (Dense)              (None, 11)                11275     
=================================================================
Total params: 40,417,099
Trainable params: 32,781,835
Non-trainable params: 7,635,264

_________________________________________________________________
```

Once the custom model is generated, test and validation images are processed using keras 'ImageDataGenerator', which is generating batches of tensor image data. The model is compiled and trained with the processed images.

Code 7. Convert Image into Tensor Image Data

```
train_dir = './train'
validation_dir = './validation'
image_size = 224
```

```
# nTrain = 600
# nVal = 150

# from keras.preprocessing.image import ImageDataGenerator
# import numpy as np
train_datagen = ImageDataGenerator(
    rescale=1./255,
    rotation_range=20,
    width_shift_range=0.2,
    height_shift_range=0.2,
    horizontal_flip=True,
    fill_mode='nearest')

validation_datagen = ImageDataGenerator(rescale=1./255)

# Change the batchsize according to your system RAM
train_batchsize = 20
val_batchsize = 20

train_generator = train_datagen.flow_from_directory(
        train_dir,
        target_size=(image_size, image_size),
        batch_size=train_batchsize,
        class_mode='categorical')

validation_generator = validation_datagen.flow_from_directory(
        validation_dir,
        target_size=(image_size, image_size),
        batch_size=val_batchsize,
        class_mode='categorical',
        shuffle=False)

# Label dictionary of train images
class_dictionary = train_generator.class_indices
print(class_dictionary)
```

Code 8. Compile and Train Model

```
# Compile the model
model.compile(loss='categorical_crossentropy',
              optimizer=optimizers.RMSprop(lr=1e-4),
              metrics=['acc'])
# Train the model
history = model.fit_generator(
    train_generator,
    steps_per_epoch=train_generator.samples/train_generator.batch_size ,
    epochs=24,
    validation_data=validation_generator,
    validation_steps=validation_generator.samples/validation_generator.batch_size,
    verbose=1)
```

```
# Save the model
model.save('faceID_VGG16.h5')
```

Accuracies

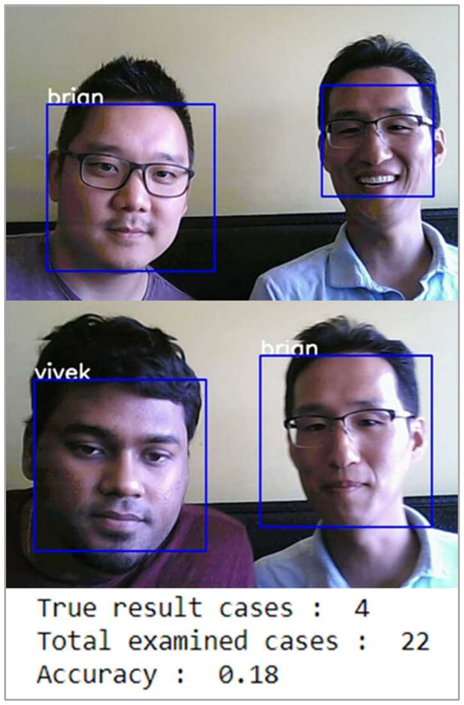
Accuracy score is used to evaluate models since it is a classification task and the differentiation between false positive and false negative predictions are not required. The accuracy equation is same as below. In short, the accuracy score measures the ratio of correct prediction out of all prediction.

$$\text{Accuracy} = \frac{True\ Positive + True\ Negative}{True\ Positive + True\ Negative + False\ Positive + False\ Negative}$$

Equation 1. Accuracy metric

We developed three models with LBPH, VGG16, and InceptionV3 algorithms. All models were trained with 33 images with 11 different people. Models were validated with 22 images with same 11 people or classes. These data sets are very small for deep learning models. For this proof of concept phase these small sets of data is okay but a final model will require large amount of data sets.

LBPH algorithm showed 18% accuracy. Below figure explains how the OpenCV program works. The blue rectangle shows the section of image that is used for prediction. The name label is shown on top of the blue rectangle. If the confidence level for a prediction is lower than a predefined threshold value, labels are not appearing on top of the blue box.



```
True result cases :  4
Total examined cases :  22
Accuracy :  0.18
```

Figure 10. LBPH Algorithm Accuracy

The accuracy scores for CNN models were higher than the LBPH models. In average from multiple runs, the VGG16 model showed 90% accuracy and the InceptionV3 models showed 60% accuracy. Thus, VGG16 model is selected as our model through this project.

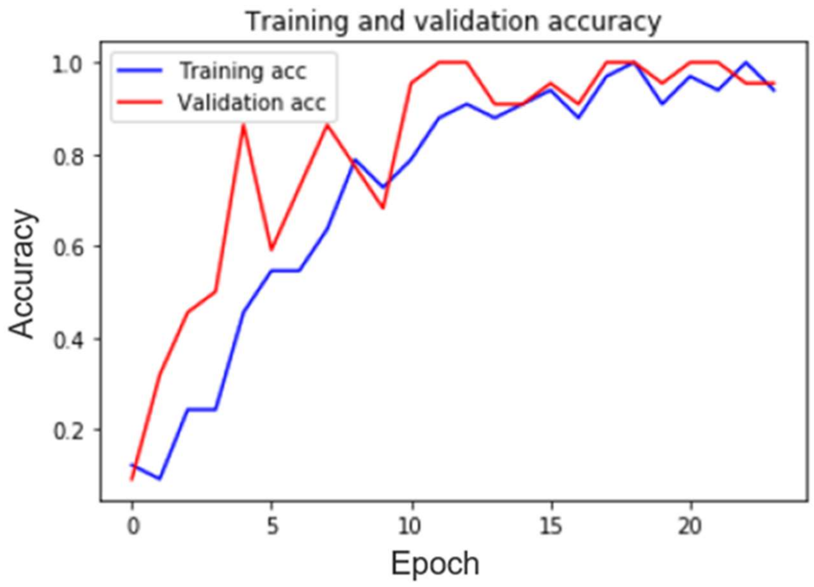


Figure mm. CNN Model Accuracy with VGG16 Architecture

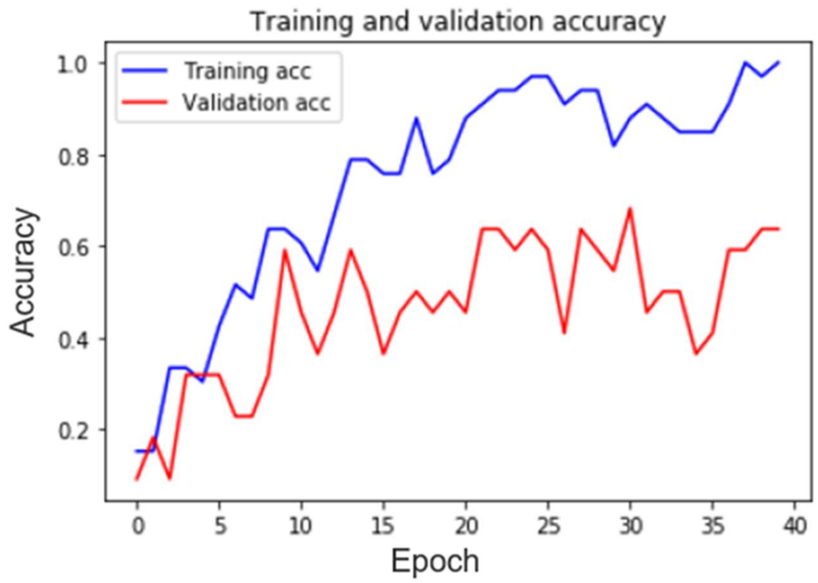
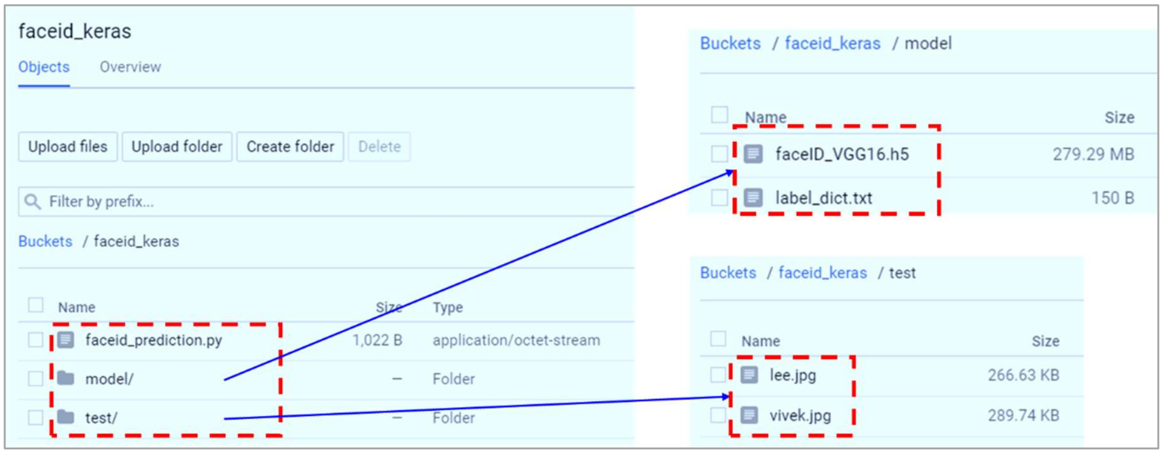


Figure 11. CNN Model Accuracy with InceptionV3 Architecture

Deploying Image Classification Model

The CNN model using the VGG16 architecture is trained locally to classify 11 people that we used for training process. We utilized google compute instance and storage bucket to deploy our model so that we can show the concept of using a prediction model from anywhere.

Prediction files are stored in a storage bucket as shown on the below figure. Top directory contains a script file, 'faceid_prediction.py' that load classification model and print out prediction results using a test image. Model directory contains a keras model and a dictionary file that will be used to look up names using prediction output numbers. In the test directory, we included two images to test our cloud model.



Buckets / faceid_keras

Name	Size	Type
faceid_prediction.py	1,022 B	application/octet-stream
model/	—	Folder
test/	—	Folder

Buckets / faceid_keras / model

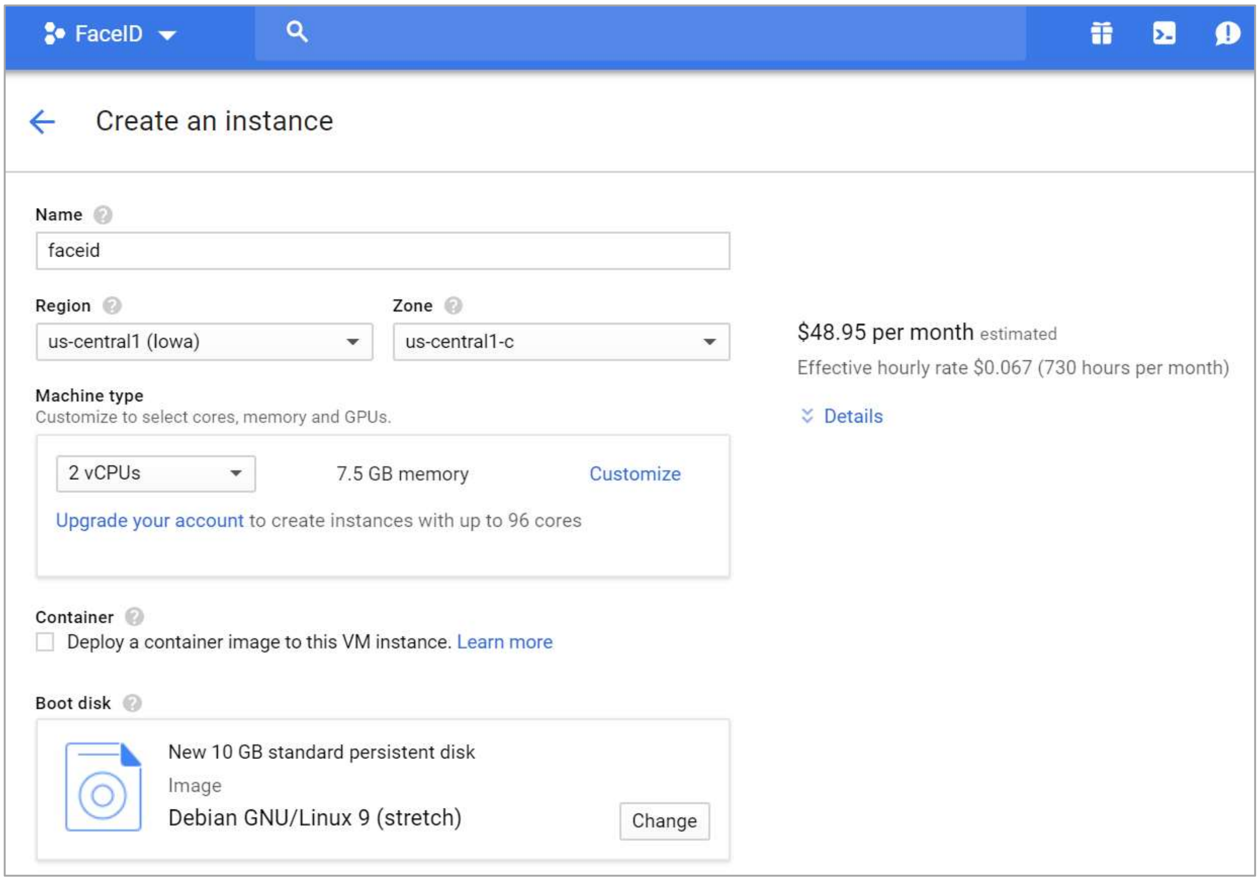
Name	Size
faceID_VGG16.h5	279.29 MB
label_dict.txt	150 B

Buckets / faceid_keras / test

Name	Size
lee.jpg	266.63 KB
vivek.jpg	289.74 KB

Figure 12. Storage Bucket Layout

Compute instance was created as shown below, and the instance was connected using a shell command window.


FaceID
Create an instance
Details
Customize
Upgrade your account to create instances with up to 96 cores
Customize to select cores, memory and GPUs.
Change

Name: faceid

Region: us-central1 (Iowa)

Zone: us-central1-c

$48.95 per month estimated

Effective hourly rate $0.067 (730 hours per month)

Machine type: 2 vCPUs, 7.5 GB memory

Container: Deploy a container image to this VM instance. Learn more

Boot disk: New 10 GB standard persistent disk

Image: Debian GNU/Linux 9 (stretch)

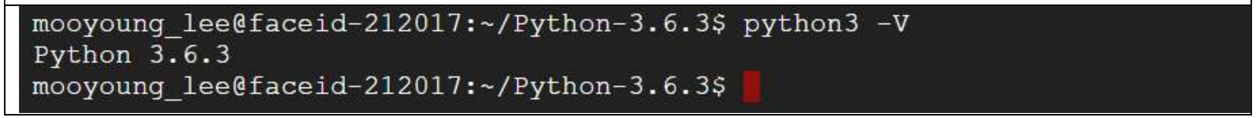
Figure 13. Google compute instance used

Python 3.6 version is installed since the version of default Python installed on the Linux instance was 2.7. The order of installing libraries are sometimes important to install Python successfully.

Code 9. Install Python 3.6.3

```
> sudo apt-get update
> sudo su
> sudo apt-get install -y libprotobuf-dev libleveldb-dev libsnappy-dev libhdf5-serial-dev protobuf-compiler
libopencv-dev
>sudo apt-get install make

> sudo apt-get install -y build-essential checkinstall
> sudo apt-get install -y libreadline-gplv2-dev libncursesw5-dev libssl-dev libsqlite3-dev tk-dev libgdbm-dev
libc6-dev libbz2-dev

wget https://www.python.org/ftp/python/3.6.3/Python-3.6.3.tgz
tar -xvf Python-3.6.3.tgz

>cd Python-3.6.3

>./configure

>sudo apt-get install zlib1g-dev

>sudo make

>sudo make install

>python3 -V
```

source :< https://stackoverflow.com/questions/47273260/google-cloud-compute-engine-change-to-python-3-6?rq=1>

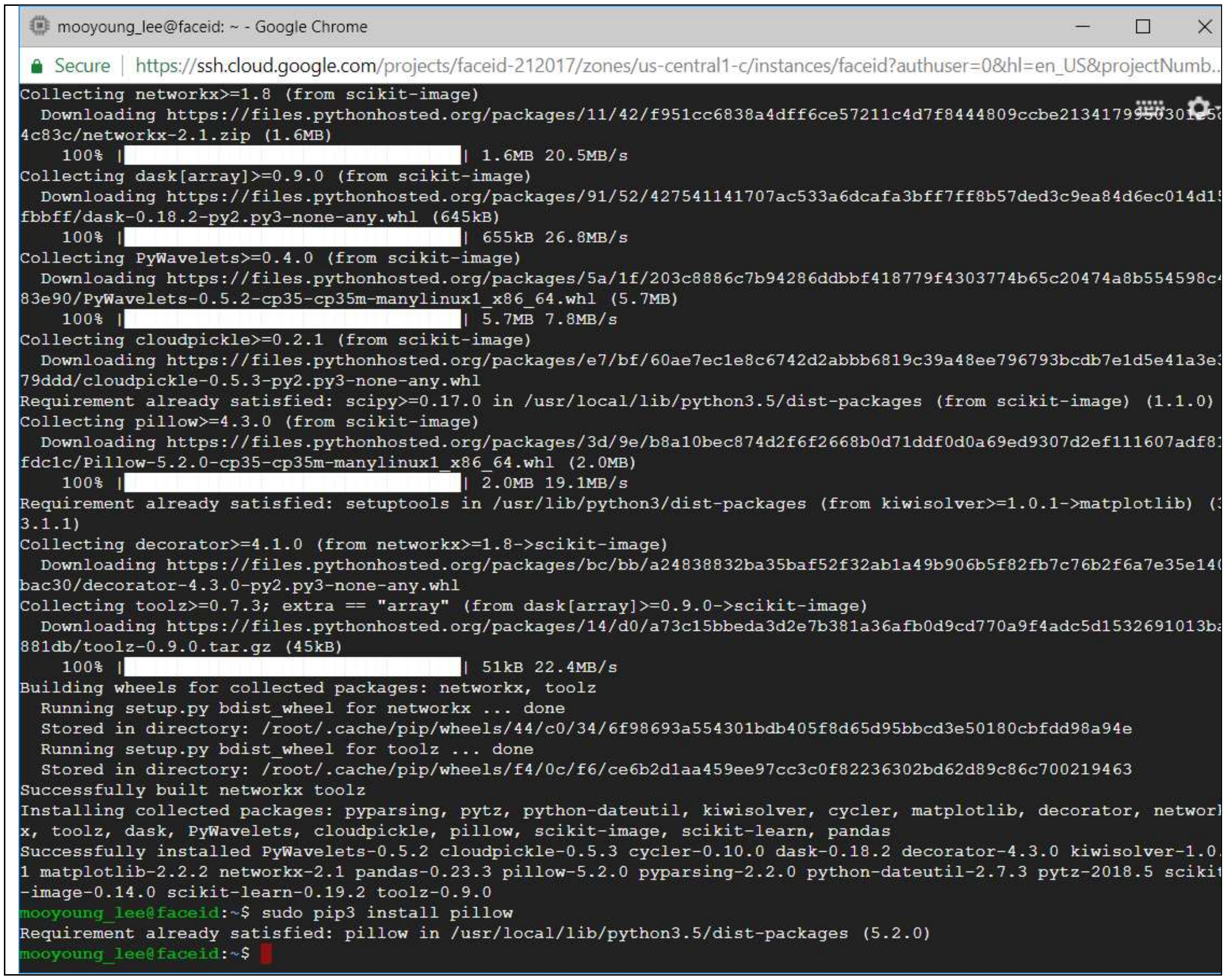


```
mooyoung_lee@faceid-212017:~/Python-3.6.3$ python3 -V
Python 3.6.3
mooyoung_lee@faceid-212017:~/Python-3.6.3$
```

Once Python 3.6 is installed, tensorflow and keras models are installed as shown below to run our deep learning model.

Code 10. Install Deep Learning Libraries

```
> sudo pip3 install tensorflow
> sudo pip3 install tensorflow  keras matplotlib scikit-image scikit-learn pandas
> sudo pip3 install pillow
```



```
mooyoung_lee@faceid: ~ - Google Chrome
Secure | https://ssh.cloud.google.com/projects/faceid-212017/zones/us-central1-c/instances/faceid?authuser=0&hl=en_US&projectNumb..
Collecting networkx>=1.8 (from scikit-image)
  Downloading https://files.pythonhosted.org/packages/11/42/f951cc6838a4dff6ce57211c4d7f8444809ccbe213417995030
4c83c/networkx-2.1.zip (1.6MB)
    100% |████████████████████████████████| 1.6MB 20.5MB/s
Collecting dask[array]>=0.9.0 (from scikit-image)
  Downloading https://files.pythonhosted.org/packages/91/52/427541141707ac533a6dcafa3bff7ff8b57ded3c9ea84d6ec014d1
fbbff/dask-0.18.2-py2.py3-none-any.whl (645kB)
    100% |████████████████████████████████| 655kB 26.8MB/s
Collecting PyWavelets>=0.4.0 (from scikit-image)
  Downloading https://files.pythonhosted.org/packages/5a/1f/203c8886c7b94286ddbbf418779f4303774b65c20474a8b554598c
83e90/PyWavelets-0.5.2-cp35-cp35m-manylinux1_x86_64.whl (5.7MB)
    100% |████████████████████████████████| 5.7MB 7.8MB/s
Collecting cloudpickle>=0.2.1 (from scikit-image)
  Downloading https://files.pythonhosted.org/packages/e7/bf/60ae7ec1e8c6742d2abbb6819c39a48ee796793bcdb7e1d5e41a3e
79ddd/cloudpickle-0.5.3-py2.py3-none-any.whl
Requirement already satisfied: scipy>=0.17.0 in /usr/local/lib/python3.5/dist-packages (from scikit-image) (1.1.0)
Collecting pillow>=4.3.0 (from scikit-image)
  Downloading https://files.pythonhosted.org/packages/3d/9e/b8a10bec874d2f6f2668b0d71ddf0d0a69ed9307d2ef111607adf8
fdc1c/Pillow-5.2.0-cp35-cp35m-manylinux1_x86_64.whl (2.0MB)
    100% |████████████████████████████████| 2.0MB 19.1MB/s
Requirement already satisfied: setuptools in /usr/lib/python3/dist-packages (from kiwisolver>=1.0.1->matplotlib) (
3.1.1)
Collecting decorator>=4.1.0 (from networkx>=1.8->scikit-image)
  Downloading https://files.pythonhosted.org/packages/bc/bb/a24838832ba35baf52f32ab1a49b906b5f82fb7c76b2f6a7e35e14
bac30/decorator-4.3.0-py2.py3-none-any.whl
Collecting toolz>=0.7.3; extra == "array" (from dask[array]>=0.9.0->scikit-image)
  Downloading https://files.pythonhosted.org/packages/14/d0/a73c15bbeda3d2e7b381a36afb0d9cd770a9f4adc5d1532691013b
881db/toolz-0.9.0.tar.gz (45kB)
    100% |████████████████████████████████| 51kB 22.4MB/s
Building wheels for collected packages: networkx, toolz
  Running setup.py bdist_wheel for networkx ... done
  Stored in directory: /root/.cache/pip/wheels/44/c0/34/6f98693a554301bdb405f8d65d95bbcd3e50180cbfdd98a94e
  Running setup.py bdist_wheel for toolz ... done
  Stored in directory: /root/.cache/pip/wheels/f4/0c/f6/ce6b2d1aa459ee97cc3c0f82236302bd62d89c86c700219463
Successfully built networkx toolz
Installing collected packages: pyparsing, pytz, python-dateutil, kiwisolver, cycler, matplotlib, decorator, networ
x, toolz, dask, PyWavelets, cloudpickle, pillow, scikit-image, scikit-learn, pandas
Successfully installed PyWavelets-0.5.2 cloudpickle-0.5.3 cycler-0.10.0 dask-0.18.2 decorator-4.3.0 kiwisolver-1.0
1 matplotlib-2.2.2 networkx-2.1 pandas-0.23.3 pillow-5.2.0 pyparsing-2.2.0 python-dateutil-2.7.3 pytz-2018.5 sciki
-image-0.14.0 scikit-learn-0.19.2 toolz-0.9.0
mooyoung_lee@faceid:~$ sudo pip3 install pillow
Requirement already satisfied: pillow in /usr/local/lib/python3.5/dist-packages (5.2.0)
mooyoung_lee@faceid:~$
```

To connect the bucket that contains our model into the compute instance that we are using, follow gcsfuse command is used. We created a new folder, 'id', and fused with the bucket, 'faceid_keras'. Once the bucket directories are fused into an instance, the bucket directories can be used as a local directory.

Code 11. Bucket-Instance Fuse

```
> mkdir id
> sudo gcsfuse faceid_keras id
```

Once the Python environment is set up and the bucket files are accessible, the classification model can be executed by running a Python script. Below code is set to look for 'test.jpg' under a specific test directory. This code can be changed to take all images in the directory. Our deep learning model is also designed to take multiple inputs at once so there is no problem taking multiple images. For future application development, the model loading and compile section can be separated from model prediction part to make system efficient. Our code is written in order to give out top 5 most probable person's name as shown below.

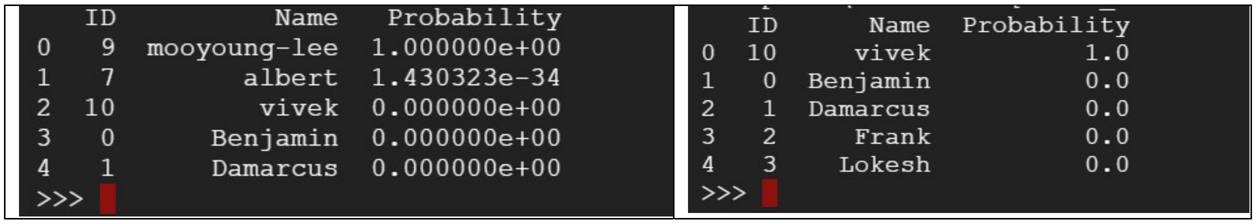
Code 12. Execution of Deep Learning Model on a Cloud Instance

```
from PIL import Image, ImageOps
```

```
image = Image.open('id/test/test.jpg')

import numpy as np
from keras.models import load_model
from keras import optimizers
import pandas as pd
import json

# test file prep
size = (224,224)
img = ImageOps.fit(image, size, Image.ANTIALIAS)
img = np.reshape(img,[1,224,224,3])

# load label
with open('id/model/label_dict.txt', 'rb') as f:
        old_labels = json.load(f)
        labels = {v:k for k, v in old_labels.items()}

# load model
model = load_model('id/model/faceID_VGG16.h5')

# compile model
model.compile(loss='categorical_crossentropy',optimizer=optimizers.RMSprop(lr=1e-4), metrics=['acc'])


# top-5 prediction
Num_Prediction = 5   # Number of people most matching
label = pd.DataFrame(list(labels.items()), columns = ['ID','Name'])
label['Probability'] = model.predict(img)[0]
label.sort_values(by=['Probability'], axis = 0, ascending = False, inplace = True)
label.reset_index(drop=True, inplace=True)
print(label.iloc[0:Num_Prediction,:])
```

The model on the google cloud is tested with two images as shown on the below figure. Both people were identified correctly. The prediction output using 'vivek.jpg' was too perfect since the image was used to train the model.

Prediction w/ "lee.jpg"	**Prediction w/ "vivek.jpg"**



```
   ID         Name   Probability
0   9  mooyoung-lee  1.000000e+00
1   7        albert  1.430323e-34
2  10         vivek  0.000000e+00
3   0      Benjamin  0.000000e+00
4   1      Damarcus  0.000000e+00
>>>
```

```
   ID      Name  Probability
0  10     vivek          1.0
1   0  Benjamin          0.0
2   1  Damarcus          0.0
3   2     Frank          0.0
4   3    Lokesh          0.0
>>>
```

Figure 14. Prediction output from a google cloud instance

Conclusions, Discussion, and Future Considerations

Convolutional neural network(CNN) models were performed better than the Local Binary Patterns Histograms (LBPH) model when identifying people using frontal face images. CNN model using VGG16 architecture showed better accuracy than the InceptionV3 architecture in our application. This conceptual demonstration should be developed further in order to be used as a real-life working product. First, the number of people for train and validation process need to be increased more than 100 people because the target application requires to classify about 100 kids. Also, the cloud model that we developed can be used as it is by using shell commands, it will be nicer if there is a user-friendly application to communicate with cloud model to make identification process faster.

The proof of concept proposed using the VGG16 image recognition model show that the ability for the proposed system is sound. Though this may be true to demonstrate that we would have the capability of performing the task, one of the major hurdles to implementation will be obtaining buy in from the leaders of the non-profit organization. Due to concerns surrounding stigma surrounding the question, "what are you going to do with this information?" there was quite a bit skepticism around the idea of this project. With continued development and clear foundational information provided to the stakeholders, it could be possible, in time, to obtain the technical information necessary to implement in their current environment, which appears to only have a webserver.

Reference

Alom, Md Zahangir et al., Improved Inception-Residual Convolutional Neural Network for Object Recognition.

Hattab, Rose. 2016. “6 Biggest Challenges Faced By Refugees And Immigrants In The US.” August 22, 2016. https://www.theodysseyonline.com/6-biggest-challenges-faced-refugees-immigrants.

Krogstad, Jens Manuel, and Jynnah Radford. 2017. “Key Facts about Refugees to the U.S.” January 30, 2017. http://www.pewresearch.org/fact-tank/2017/01/30/key-facts-about-refugees-to-the-u-s/.

“Love Is Ministrity.” 2018. 2018. https://loveisministry.org/.

Qassim, Hussam, Verma, Abhishek & Feinzimer, David, Compressed residual-VGG16 CNN model for big data places image recognition. In Computing and Communication Workshop and Conference (CCWC), 2018 IEEE 8th Annual. IEEE, pp. 169–175.

Sinno Jialin Pan & Qiang Yang, A Survey on Transfer Learning. Knowledge and Data Engineering, IEEE Transactions on, 22(10), pp.1345–1359.

Solis, Dianne. 2018. “Dallas Caseworker Develops App to Help Refugees Find Their Way in New Cities.” January 5, 2018. https://www.dallasnews.com/news/immigration/2018/01/05/tools-helps-refugees-find-way-around-new-surroundings.

Zhang, Qingchen et al., A survey on deep learning for big data. Information Fusion, 42, pp.146–157.