VIVA : Installation guide

Section 1 : Environment

install Brew **https://brew.sh/**

$ ruby -e "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/master/install) "
Add the last line of ~/.profile file to export PATH=/usr/local/bin:/usr/local/sbin:$PATH to

install Python

Brew install python to

install Virtual Environment and Tensorflow **https://www. Tensorflow.org/install/install_mac**

$ sudo easy_install pip # Install pip manager
$ pip install --upgrade virtualenv # If you get an error, you need to install nose and tornado
$ pip install nose
$ pip install tornado
$ virtualenv --system-site-packages ./tensorflow # for Python 2.7
$ virtualenv --system-site-packages -p python3 ./tensorflow # for Python 3.n
$ cd tensorflow
$ source ./bin/activate
(tensorflow)$ pip install --upgrade tensorflow # for Python 2.7
(tensorflow)$ pip3 install --upgrade tensorflow # for Python 3.nInstall

Keras https://keras.io/#installation
$keras Pip install

Tools to save Keras model saved to disk
$ brew install hdf5
$ pip install h5py

Install Pillow
$ pip install pillow

Install XCode
to the web to download and install XCode https://developer.apple.com/xcode/ Or App-Store download
$ sudo xcode-select --install
$ sudo xcodebuild -license # Swipe to the bottom and accept the terms

install OPENCV https://www.learnopencv.com/install-opencv3-on-macos/
$ brew Install opencv

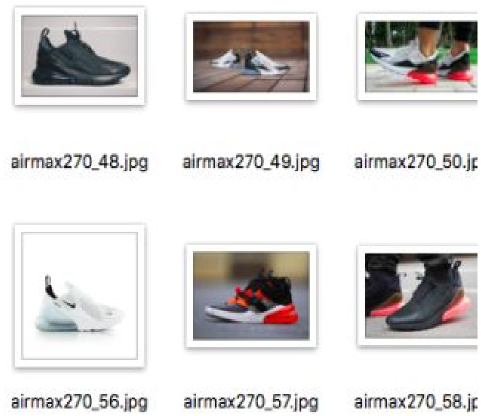
if there is a permission problem e.g. brow link isl, brew link gcc, brew link hdf5
$ sudo chown -R ***myaccount***: admin /usr/local/bin e.g. sudo chown -R **eddieliu**: admin/usr/local/bin
$ sudo chown -R ***myaccount:*** admin /usr/local/share e.g. sudo chown -R **eddieliu:** admin/usr/local/share

Section 2: Data Collection

Good data is a very critical. The quality of model highly depends on training data.

Size: at lease 300x300
sheets: at least 100 pictures per category.
Format: JPG

image selection
- Front product photo
- Real world picture with backgrounds
- Various angles
- Various brightness



When doing machine learning, you can imagine you are asking the machine to learn and tell the difference between categories. The algorithm will define categories according to the content of the picture.

For example, category : Jeans

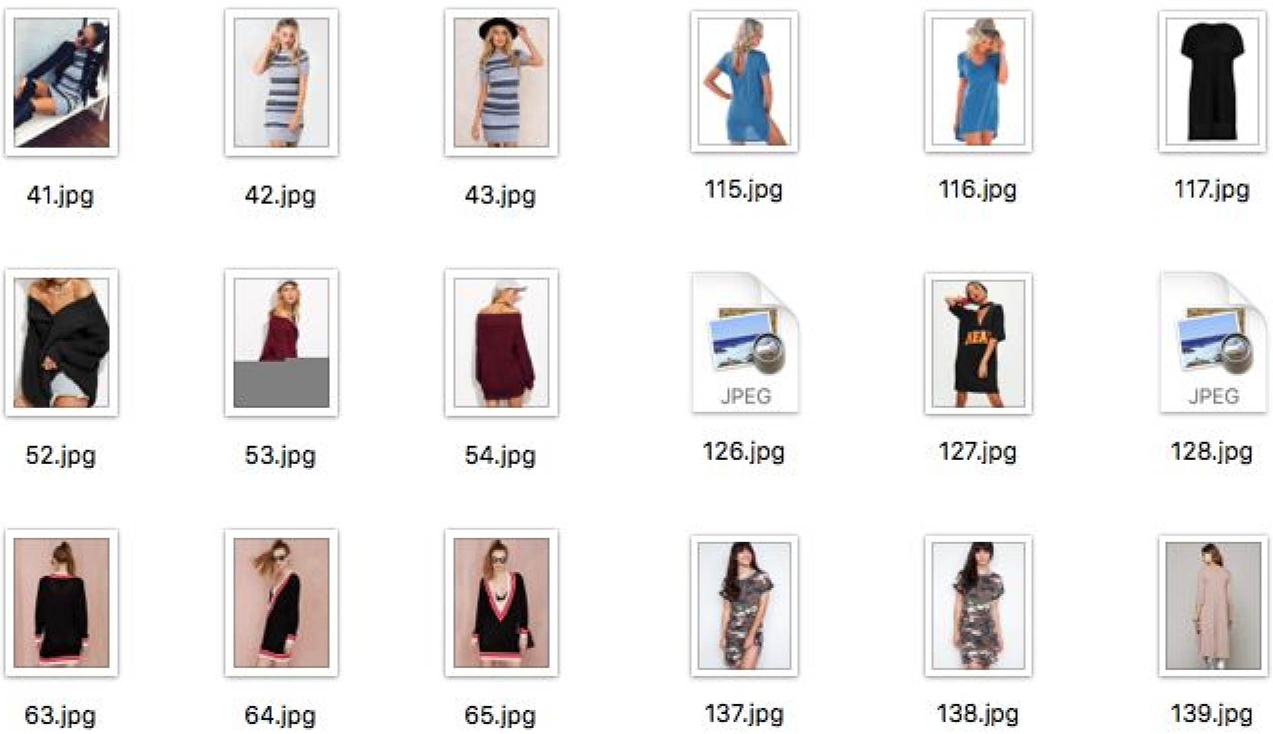
(√) These are good picture choices, jeans are the main content
The machine learns that long, various blue fabric and with or without a pair of shoes underneath, are all jeans.

(x) These are relatively not that good picture choice, because those pictures cover a lot of other elements such as upper body, head, shoes, etc. The machine may conclude that all those parts are necessary elements of "jeans".

Clean Data

When you download a large amount of data from the internet, sometimes the file is corrupted and you must do the double check. Otherwise, when processing error pictures, the program will stop. The following 53.jpg, 126.jpg, 128.jpg, are all corrupted problematic. Be sure to delete, or re-download those images



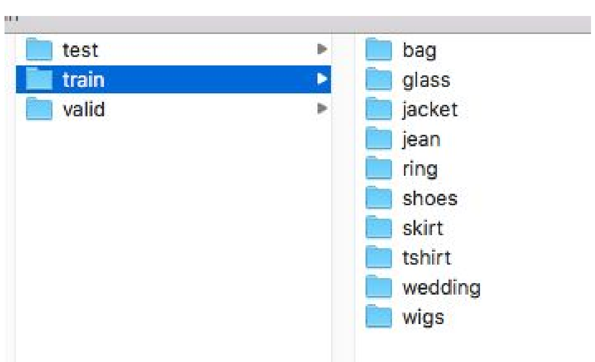
Folder structure.

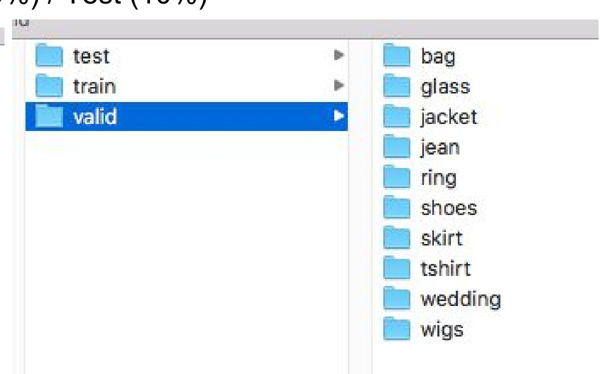
Train : put training data in relative subfolders. The program will train the model from the dataset

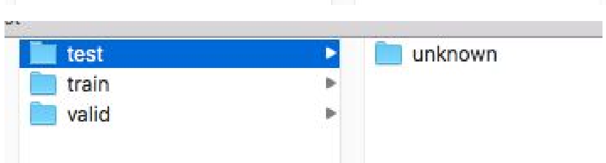
Valid: put validating data in relative subfolders. The program will validate the trained model from the dataset

Test : testing data in the subfolder "unknown." The program will predict the categories of the dataset

Ratio recommended : Train (80%) / Valid(10%) / Test (10%)







Section 3 : Train Model

1. Change parameters in config.json

```
{
 "train_path" :"dataset",
 "batch_size_train":"15",
 "category_num": "3",
 "learning_rate":"0.0003",
 "epoch_num":"20",
 "steps_per_epoch":"6",
 "output_name":"model.h5"
}
```

Parameter	Description	Suggested ranges
train_pathbatch	Root folder to put “Train”, “Valid”, and “Test” folders	
batch_size_train	How many pictures are trained in each batch.	20 ~ 30
category_num	How many categories we want to train	
learning_rate	How much we are adjusting the weights	0.001 ~ 0.00001
epoch_num	An epoch is an iteration over the entire dataset and updated the weight of the model	100 ~ 200
steps_per_epoch	Total number of steps (batches of samples) before declaring one epoch finished and starting the next epoch	
output_name	The name of the output file	xxxxx.h5

How long does it take to do the training ? The time is relative to the number of pictures to be processed, which means “batch size”, “steps per epoch”, and “epoch”

- number of pictures that each epoch will process = **batch_size_train * steps_per_epoch**
- **total time required for training: epoche_num * time per epoch**

epoch_num	batch_size_train	steps_per_epoch	total time
10	**10**	**10**	**1 T**
10	**20**	**10**	**2 T**
10	**10**	**20**	**2 T**
10	**15**	**20**	**3 T**
80	**15**	**20**	**24 T**

2. Command line: python train_model.py

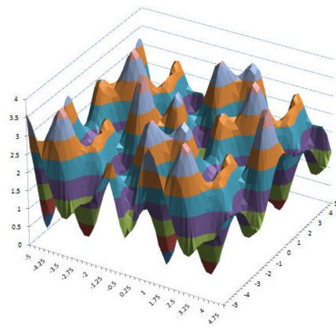
```
#!/usr/bin/env python
# -*- coding: utf-8 *

import numpy as np
import keras
from keras import backend as K
from keras.layers.core import Dense, Activation
from keras.optimizers import Adam
from keras.metrics import categorical_crossentropy
from keras.preprocessing.image import ImageDataGenerator
from keras.preprocessing import image
from keras.models import Model
from keras.applications import imagenet_utils
import json

# Open config.json
def jsonReader():
   with open("./config.json", 'r') as load_f:
       jsonDict = json.load(load_f)
       return jsonDict

# Read data from config.json
jsonData = jsonReader()
train_path = jsonData['train_path'] + '/train'
valid_path = jsonData['train_path'] + '/valid'
test_path = jsonData['train_path'] + '/test'
bsize = int(jsonData['batch_size_train'])
catnum = int(jsonData['category_num'])
lrate = float(jsonData['learning_rate'])
steps = int(jsonData['steps_per_epoch'])
epoch_num = int(jsonData['epoch_num'])
output_name = jsonData['output_name']

# Train the model. Default shuffle = true
train_batches =
ImageDataGenerator(preprocessing_function=keras.applications.mobilenet.preproces
s_input).flow_from_directory( train_path, target_size=(224,224),
batch_size=bsize)

# Validate the model.  Default shuffle = true
valid_batches =
ImageDataGenerator(preprocessing_function=keras.applications.mobilenet.preproces
s_input).flow_from_directory( valid_path, target_size=(224,224), batch_size=10)

# Make prediction with the model trained
```

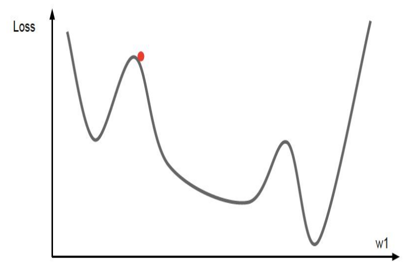
```
test_batches =
ImageDataGenerator(preprocessing_function=keras.applications.mobilenet.preproces
s_input).flow_from_directory(test_path, target_size=(224,224), batch_size=2,
shuffle=False)

mobile = keras.applications.mobilenet.MobileNet()

x = mobile.layers[-6].output
predictions = Dense(catnum, activation='softmax')(x)
model = Model(inputs=mobile.input, outputs=predictions)

model.summary()

# Only train the last 5 layers and make the previous layers fixed
for layer in model.layers[:-5]:
   layer.trainable = False
model.compile(Adam(lr=lrate), loss='categorical_crossentropy',
metrics=['accuracy'])
model.fit_generator(train_batches, steps_per_epoch=steps,
                    validation_data=valid_batches, validation_steps=2,
epochs=epoch_num, verbose=2)

# Output the model
model.save(output_name)

# Make prediction with the model
predictions = model.predict_generator(test_batches,steps=1, verbose=2)
print(predictions)
print(train_batches.class_indices)
```

Section 4 : Fine Tune Model

A good model can predict the real world results with minimum difference. During the training process, pay attention to loss and valid_acc (the accuracy in the valid set)

Imagine we are in a valley, and our goal is to find the lowest altitude (the smallest loss). The learning rate is how far we move each step.





Assume the lowest area is 5.25 km in front of us. The following are what happens with different learning rates

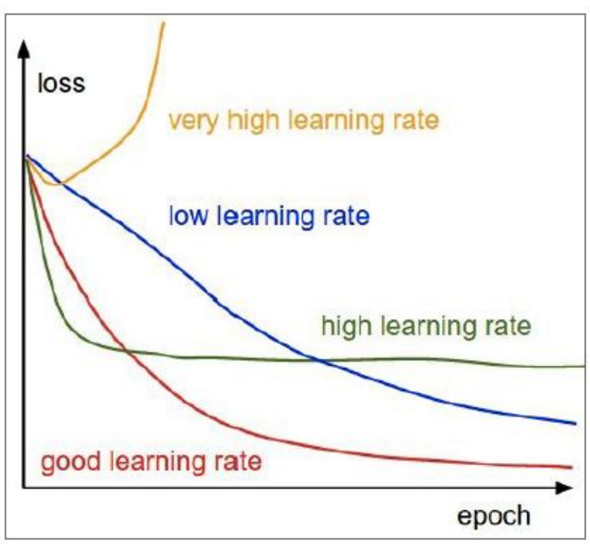
Very high learning rate : Every step is 10 km and will take us away from the minimum.

High learning rate : 1 km per step.In the early stages, loss rate dropped quickly, but there is a bottleneck, since we finally are will explore between 5 ~ 6 km.

Good learning rate : 0.1 km per step, in the early stages, **loss drops rapidly, and finally stabilizes**, and we will explore around 5.2 ~ 5.3 km

Low learning rate : Each step is 0.001 km, the loss rate drops very slowly, and it takes us a lot of time to progress.

Note: The picture trained in each batch the sequences are randomly assigned, but the training process will follow the parameters in config.json. Even if the initial configuration is the same, the result would vary.



Let's look at the training process when choosing different learning rates

<table>
<tr><td colspan="2" align="center">Very high learning rate : 10</td></tr>
<tr><td colspan="2" align="center">Loss instantly go over 15, and we can stop training</td></tr>
<tr><td colspan="2">Epoch 1/100
 - 51s - loss: 14.0048 - acc: 0.0537 - val_loss: 16.1181 - val_acc: 0.0000e+00
Epoch 2/100
 - 46s - loss: 14.9361 - acc: 0.0733 - val_loss: 16.1181 - val_acc: 0.0000e+00
Epoch 3/100
 - 43s - loss: 15.3122 - acc: 0.0500 - val_loss: 16.1181 - val_acc: 0.0000e+00
Epoch 4/100
 - 46s - loss: 14.8824 - acc: 0.0767 - val_loss: 16.1181 - val_acc: 0.0000e+00
Epoch 5/100
 - 43s - loss: 14.8824 - acc: 0.0767 - val_loss: 16.1181 - val_acc: 0.0000e+00
Epoch 6/100
 - 41s - loss: 14.7749 - acc: 0.0833 - val_loss: 16.1181 - val_acc: 0.0000e+00
Epoch 7/100
 - 44s - loss: 14.9898 - acc: 0.0700 - val_loss: 16.1181 - val_acc: 0.0000e+00
Epoch 8/100
 - 42s - loss: 14.7212 - acc: 0.0867 - val_loss: 16.1181 - val_acc: 0.0000e+00
Epoch 9/100
 - 42s - loss: 15.0436 - acc: 0.0667 - val_loss: 16.1181 - val_acc: 0.0000e+00
Epoch 10/100
 - 42s - loss: 14.8824 - acc: 0.0767 - val_loss: 16.1181 - val_acc: 0.0000e+00</td></tr>
<tr><td colspan="2" align="center">Next step: greatly reduce the learning rate</td></tr>
</table>

<table>
<tr><td colspan="2" align="center">High learning rate: 0.01</td></tr>
<tr><td>During the first 10 steps, the loss quickly drops from 2.7 to 0.5 and 7.1 to 2.1, and accuracy also quickly reached 0.7~0.8 and 0.3~0.4</td><td>The valid accuracy reached the bottleneck 0.4~0.5. Note: The loss of this train is less than 0.1, and the train accuracy is close to 1.0, indicating the model is “overfit.” It fits too much with the training data, but the accuracy for prediction is bad.</td></tr>
<tr><td>Epoch 1/100
 - 42s - loss: 2.7851 - acc: 0.1833 - val_loss: 7.1870 - val_acc: 0.0000e+00
Epoch 2/100
 - 38s - loss: 2.1982 - acc: 0.3008 - val_loss: 5.9909 - val_acc: 0.2000
Epoch 3/100
 - 42s - loss: 2.0329 - acc: 0.3667 - val_loss: 3.1621 - val_acc: 0.4000
Epoch 4/100
 - 41s - loss: 1.6221 - acc: 0.4867 - val_loss: 3.8205 - val_acc: 0.2000
Epoch 5/100
 - 39s - loss: 1.5566 - acc: 0.4600 - val_loss: 6.1969 - val_acc: 0.0000e+00
Epoch 6/100
 - 37s - loss: 1.4096 - acc: 0.5400 - val_loss: 5.0262 - val_acc: 0.2000
Epoch 7/100
 - 37s - loss: 0.7210 - acc: 0.7500 - val_loss: 5.4693 - val_acc: 0.0000e+00
Epoch 8/100
 - 38s - loss: 0.6067 - acc: 0.8333 - val_loss: 3.3162 - val_acc: 0.2000
Epoch 9/100
 - 37s - loss: 0.4823 - acc: 0.8767 - val_loss: 2.9067 - val_acc: 0.4000
Epoch 10/100
 - 36s - loss: 0.5013 - acc: 0.8733 - val_loss: 2.1334 - val_acc: 0.3000</td><td>Epoch 81/100
 - 38s - loss: 0.0121 - acc: 1.0000 - val_loss: 2.5431 - val_acc: 0.4000
Epoch 82/100
 - 47s - loss: 0.0113 - acc: 1.0000 - val_loss: 2.3135 - val_acc: 0.3000
Epoch 83/100
 - 44s - loss: 0.0086 - acc: 1.0000 - val_loss: 2.4654 - val_acc: 0.4000
Epoch 84/100
 - 45s - loss: 0.0166 - acc: 0.9967 - val_loss: 2.8840 - val_acc: 0.5000
Epoch 85/100
 - 39s - loss: 0.0105 - acc: 1.0000 - val_loss: 2.6925 - val_acc: 0.5000
Epoch 86/100
 - 38s - loss: 0.0083 - acc: 1.0000 - val_loss: 2.1870 - val_acc: 0.4000
Epoch 87/100
 - 39s - loss: 0.0071 - acc: 1.0000 - val_loss: 2.2050 - val_acc: 0.4000
Epoch 88/100
 - 37s - loss: 0.0080 - acc: 1.0000 - val_loss: 2.2907 - val_acc: 0.4000
Epoch 89/100
 - 39s - loss: 0.0068 - acc: 1.0000 - val_loss: 2.2843 - val_acc: 0.4000</td></tr>
<tr><td colspan="2" align="center">Next : reduce the learning rate</td></tr>
</table>

Low learning rate: 0.00001	
The loss decreases slowly	The valid accuracy increase very slow
Epoch 1/100 - 41s - loss: 3.2658 - acc: 0.0867 - val_loss: 2.9313 - val_acc: 0.2000 Epoch 2/100 - 39s - loss: 3.3687 - acc: 0.0767 - val_loss: 2.9106 - val_acc: 0.2000 Epoch 3/100 - 42s - loss: 3.1668 - acc: 0.0667 - val_loss: 2.8740 - val_acc: 0.2000 Epoch 4/100 - 43s - loss: 3.1335 - acc: 0.0900 - val_loss: 2.8356 - val_acc: 0.2000 Epoch 5/100 - 42s - loss: 3.1657 - acc: 0.0800 - val_loss: 2.8029 - val_acc: 0.2000 Epoch 6/100 - 38s - loss: 3.1580 - acc: 0.0872 - val_loss: 2.7779 - val_acc: 0.2000 Epoch 7/100 - 43s - loss: 2.9513 - acc: 0.0667 - val_loss: 2.7578 - val_acc: 0.2000 Epoch 8/100 - 44s - loss: 2.9811 - acc: 0.0900 - val_loss: 2.7353 - val_acc: 0.2000 Epoch 9/100 - 42s - loss: 2.9064 - acc: 0.1100 - val_loss: 2.7239 - val_acc: 0.2000 Epoch 10/100 - 41s - loss: 2.8591 - acc: 0.1133 - val_loss: 2.7067 - val_acc: 0.2000	Epoch 91/100 - 37s - loss: 1.4415 - acc: 0.5930 - val_loss: 1.9144 - val_acc: 0.3000 Epoch 92/100 - 40s - loss: 1.4591 - acc: 0.6167 - val_loss: 1.9073 - val_acc: 0.3000 Epoch 93/100 - 43s - loss: 1.4684 - acc: 0.5867 - val_loss: 1.9024 - val_acc: 0.3000 Epoch 94/100 - 42s - loss: 1.4227 - acc: 0.5800 - val_loss: 1.9028 - val_acc: 0.3000 Epoch 95/100 - 43s - loss: 1.4395 - acc: 0.5800 - val_loss: 1.9013 - val_acc: 0.3000 Epoch 96/100 - 41s - loss: 1.3721 - acc: 0.6267 - val_loss: 1.8974 - val_acc: 0.3000 Epoch 97/100 - 42s - loss: 1.4560 - acc: 0.6033 - val_loss: 1.8928 - val_acc: 0.3000 Epoch 98/100 - 43s - loss: 1.4318 - acc: 0.5700 - val_loss: 1.8878 - val_acc: 0.3000 Epoch 99/100 - 42s - loss: 1.4992 - acc: 0.5982 - val_loss: 1.8804 - val_acc: 0.3000 Epoch 100/100 - 37s - loss: 1.3924 - acc: 0.6100 - val_loss: 1.8779 - val_acc: 0.3000
Next: slightly enhance	

Good Learning Rate	
The first 10 steps, loss decreased from 3 to 1.8	The last 10 steps, loss are and accuracy are stabalized within satisfactory range
Epoch 1/100 - 39s - loss: 3.0220 - acc: 0.0967 - val_loss: 2.5357 - val_acc: 0.2000 Epoch 2/100 - 36s - loss: 2.8572 - acc: 0.1233 - val_loss: 2.5735 - val_acc: 0.2000 Epoch 3/100 - 36s - loss: 2.6347 - acc: 0.1533 - val_loss: 2.5309 - val_acc: 0.2000 Epoch 4/100 - 36s - loss: 2.4472 - acc: 0.2000 - val_loss: 2.5096 - val_acc: 0.2000 Epoch 5/100 - 37s - loss: 2.4605 - acc: 0.1667 - val_loss: 2.4614 - val_acc: 0.2000 Epoch 6/100 - 45s - loss: 2.3777 - acc: 0.2151 - val_loss: 2.3947 - val_acc: 0.3000 Epoch 7/100 - 37s - loss: 2.1120 - acc: 0.3127 - val_loss: 2.3821 - val_acc: 0.2000 Epoch 8/100 - 42s - loss: 2.0617 - acc: 0.3500 - val_loss: 2.3293 - val_acc: 0.3000 Epoch 9/100 - 43s - loss: 1.9819 - acc: 0.3667 - val_loss: 2.3203 - val_acc: 0.4000 Epoch 10/100 - 1136s - loss: 1.8970 - acc: 0.4033 - val_loss: 2.2391 - val_acc: 0.3000	Epoch 91/100 - 37s - loss: 0.4574 - acc: 0.9426 - val_loss: 1.2659 - val_acc: 0.7000 Epoch 92/100 - 43s - loss: 0.3990 - acc: 0.9633 - val_loss: 1.2293 - val_acc: 0.8000 Epoch 93/100 - 42s - loss: 0.4320 - acc: 0.9600 - val_loss: 1.2142 - val_acc: 0.8000 Epoch 94/100 - 42s - loss: 0.3907 - acc: 0.9667 - val_loss: 1.2125 - val_acc: 0.8000 Epoch 95/100 - 39s - loss: 0.3538 - acc: 0.9767 - val_loss: 1.2097 - val_acc: 0.8000 Epoch 96/100 - 42s - loss: 0.4138 - acc: 0.9600 - val_loss: 1.2195 - val_acc: 0.7000 Epoch 97/100 - 43s - loss: 0.4060 - acc: 0.9598 - val_loss: 1.2310 - val_acc: 0.7000 Epoch 98/100 - 47s - loss: 0.3656 - acc: 0.9700 - val_loss: 1.2328 - val_acc: 0.7000 Epoch 99/100 - 42s - loss: 0.3816 - acc: 0.9667 - val_loss: 1.2316 - val_acc: 0.8000 Epoch 100/100 - 43s - loss: 0.3710 - acc: 0.9733 - val_loss: 1.2435 - val_acc: 0.7000
next step:it is suitable for the learning rate, only the fine tuning, can be higher to see whether to allowAccuracy:	

Section 5: TFJS converter (Keras h5 turn TFJS)

reference: [https ://github.com/tensorflow/tfjs-converter](https://github.com/tensorflow/tfjs-converter)

install Tensorflowjs

pip install tensorflowjs

convert format

```
tensorflowjs_converter \
    --input_format=keras \
    /tmp/my_keras_model.h5 \
    /tmp/my_tfjs_model
```

Section 6: Integration With Front-End Web application

GitHub : https://github.com/pdaexample/TFJS_Example

1. Upload converted model to the server. Remember the path of the **model.json**
2. **Create MobileNet Object.** The object includes “load model”, “predict”, and link the results with business flow

```
var MobileNet = {
  / * TODO: load the url of the model * /
      MODEL_URL: 'https://your.model.url',
  /* The DIVISOR is a fixed value. DO NOT modify */
  PREPROCESS_DIVISOR: 127.5.
  /* Accept when probability of prediction is more than or equal to */
  PICKUP_RATE: 0.7,
  constructor: function (){},
  /* TODO(1): check whether the model is loaded */

  /* TODO(2): check the network environment */

  /* Load model */
  load: async Function (){
    This= await .modeltf.loadModel(this.MODEL_URL);
    /* The first simulation after loading the model, which can speed up the
first prediction */
    This.predict(tf.zeros([1, 224, 224, 3])).dispose();
    /* Bind camera after loading. You can add your own business flow /
    camera_ops.setup ('detector ', 'video', 'closeCamera ', 'outputCanvas ',
'camera ');
  },
  /* Clear the cache after prediction */
  dispose: function () {
    This.model.dispose();
  },
  /* Predict function. Normalize the image and make prediction */
  predict: function (screenshot) {
    Console.time();
    Const offset = tf.scalar(this.PREPROCESS_DIVISOR);
    Const nomalized = screenshot.sub(offset).div(offset);
    Const batched = nomalized.reshape([1,224,224,3]);
    Return This.model.predict(batched);
  },
  /* Customize the predicted result with your business flow */
   getTopResult: async function (logits) {
    /*  Get the value of result */
```

```
    Const values = await logits.data();
    /* Pass the result to the array */
    Const valuesAndIndices = [];
    For (let i = 0; i < values.length; i++) {
      valuesAndIndices.push({value: values[i], index: i});
    }
    /* Sort */
    valuesAndIndices.sort((a, b) => {
      Return b.value - a.value;
    });

    /* TODO: link the result with business flow */

    Console.log(valuesAndIndices);
    Console.timeEnd();

  }
}
```

3. **Use PWA function to check the user's condition**
    a. **Whether to support Service Worker**

```
if ('serviceWorker' In navigator) {
       Window.addEventListener('load', function() {
         navigator.serviceWorker.register('/sw.js').then(function(registration) {
           // Registration was successful
           Console.log('ServiceWorker registration successful with scope: ',
registration.scope);
         }, function(err) {
           // registration failed :(
           Console.log('ServiceWorker registration failed: ', err);
         });
         });
       }
```

    b. **Has the model been loaded: TODO(1)**

```
check_if_model_exist: function () {
    caches.has('https://your.model. Url').then(function (result){
      Return result;
    });
  }
```

    c. **If the model is not loaded, check if the device is in WIFI environment: TODO(2)**

```
check_isWifi: function() {
```

```
    If (!navigator.connection || navigator.connection.type != "wifi") {
      Return False;
    }
    Return True;
  }
```

Combined :

```
if(!MobileNet.check_if_model_exist()) {
  If (MobileNet.check_isWifi()){
    MobileNet.load();
  }
} else {
  MobileNet.load();
}
```

4. **Include Tensorflow JS library on HTML**

```
<script src="https://cdn.jsdelivr.net/npm/
@tensorflow/tfjs@0.12.0/dist/tf.min.js"></script>
<script src="https://cdn.jsdelivr.net/npm/@tensorflow/tfjs-converter@0.5.0/
Dist/tf-converter.min.js"></script>
```

5. **Add camera function and camera layer inside HTML body**

```
<div id="cameraLayer" style="display:none;">
  <i class="material-icons" id="closeButton" style="display:none;">close</i>
  <video id="video" playinline="playinline" style="display:none;"></video>
</div>
<canvas id="middleCanvas" style="display:none;"></canvas>
<id=Img"imageToPredict" style="display:none;" src="" width="224" height="224" />
<div id="loadingLayer"><img id="loading" src="/images /loading.gif" /></div>
```

6. **Bind the camera to trigger the prediction**

```
/* Set the camera object to trigger the prediction of images */
var camera_ops = {
  IDENTIFY_WAIT_TIME: 1500, // Wait time for capturing images
  Take thecameraLayer: undefined, / / Camera layer
  video: undefined, // Camera video DOM
  closeButton: undefined, // Close the camera button
  middleCanvas: undefined, // The middle layer to capture video and convert it
to image
  mainContent: undefined, // The main image layer
```

```
  videoStream: undefined, // video stream
  loadingLayer: undefined, // loading effect when start prediction
  /* Bind camera event */
  setupCamera: function(cameraLayerId, videoId, closeButtonId, middleCanvasId,
cameraTriggerId) {
    This.cameraLayer = document.getElementById(cameraLayerId);
    This.video = document.getElementById(videoId);
    This.closeButton = document.getElementById(closeButtonId);
    This.middleCanvas = document.getElementById(middleCanvasId);
    // TODO: Change the load layer ID
    This.loadingLayer = document.getElementById('loadingLayer')
    // TODO: Change the content of main layer Class
    This.mainContent = document.getElementsByClassName('yourMainContentClass');
    //setup the camera trigger
    This.addCameraEvent(cameraTriggerId);
    This.bindCameraLayerClose();
  },
  /* Check if the browser supports camera functions */
  checkAvailable: function () {
    If (!navigator.mediaDevices || !navigator.mediaDevices.getUserMedia) {
      Return False;
    }
    Return True;
  },
  /* Add camera event on the camera button */
  AttachaddCameraEvent: function (id) {
    Const w = screen.width;
    Const h = screen.width;
    Console.log('width:' + w + ", h:" + h);
    Const cameraButton = document.getElementById(id);

    cameraButton.addEventListener('click', async Function () {
      /* OPTIONAL TODO: Can add other conditions after camera event*/

      /* Check if the browser supports camera function*/
      If (!camera_ops.checkAvailable()) {
        general_ops.showMessage('You browser not support browser camera api');
        Return;
      }

      camera_ops.video.width = w;
      camera_ops.video.height = h;
      /* Set the camera's parameters which can be changed, please refer to:
```

```
https://developer.mozilla.org/en-US/docs/Web/API/MediaStreamConstraints */
      camera_ops.videoStream = await navigator.mediaDevices. getUserMedia({
        audio: false,
        video :{
          facingMode: 'environment',
          width: { min: 224, ideal: 448, max: 1344 },
          height: { min: 224, ideal: 448, max: 1344 }
        }
      }) ;
      /* Import the stream into the video object */
      camera_ops.video.srcObject = camera_ops.videoStream;

      Return (new Promise((resolve) => {
          /* Confirm the video stream is properly loaded */
          camera_ops.video.onloadedmetadata = () => {
            camera_ops.mainContent[0].style = 'display: none';
            camera_ops.cameraLayer.style.display = 'block';
            camera_ops.video.play ();
            camera_ops.video.style.display = 'block';
            camera_ops.closeButton.style.display = 'block';
            Console.time();
            resolve('init');

          };
        })).then(function() {
          /* Time to wait for capturing camera image */
          setTimeout(function () {
            camera_ops.loadingLayer.classList.add('show');
            This.video.pause();
            camera_ops.snapshot(); },
            camera_ops.IDENTIFY_WAIT_TIME);

        });
    })

    cameraButton.classList.add('fadeIn');
  },
  /* Close camera layer */
  bindCameraLayerClose: function () {
    This.closeButton.addEventListener('click', function () {
      camera_ops.mainContent[0].style.display = 'block';
      camera_ops.cameraLayer.style.display = 'none';
      camera_ops.video.pause();
```

```
      camera_ops .videoStream.getTracks()[0].stop();
      camera_ops.video.style.display = 'none';
      camera_ops.closeButton.style.display = 'none';
      camera_ops.loadingLayer.className = '';
    });
  },
  /* Capture the image into Canvas, convert it into a picture object, and import
the model to make the prediction */
  snapshot: function () {
    Console.timeEnd();
    Const imageToPredict = document.getElementById('imageToPredict');
    This.middleCanvas.width = camera_ops.video.width;
    This.middleCanvas.height = camera_ops.video.height;
    Let canvasContext = this.middleCanvas.getContext('2d');
    canvasContext.drawImage(camera_ops.video, 0, 0, camera_ops.video.width,
camera_ops.video.height);
    imageToPredict.src = this.middleCanvas.toDataURL(' Image/jpeg');
    imageToPredict.width = '224';
    imageToPredict.height = '224';
    imageToPredict.onload = function ()
{predict_ops.runPredict(imageToPredict);}
  }
};
```

7. **Link camera and the MobileNet module**

```
/* link Camera and the MobileNet module*/
var predict_ops = {
  runPredict: function (input) {
    /* Convert it to TensorFlow JS compatible image format */
    Const pixels = tf.fromPixels(input).toFloat();
    Let result = MobileNet.predict(pixels);
    MobileNet.getTopResult(result);
  }
}
```

The document is credit to gTech Velocity MSC CN team (Eddie Liu, Palances Liao, Keith Gu, Cecilia Cong, Victor Shen), and advices from CN TensorFlow team (Tiezhen Wang)