# Computer Vision Workshop

## learn to code AI powered apps

Shu Ishida – Oxford Microsoft Student Partners

## Contents

# Introduction

## With Computer Vision, you can solve many problems

What can you do if your app can understand what is happening in the world? Computer Vision is perfect for *automated tasks, IoT devices, monitoring, augmented reality,* and detecting all kinds of visual signals (facial expressions, gestures, etc.).

Equipped with computer vision, you can start programming a robot to navigate around the room, search through your album collection based on specific features, or set up a web cam on your laptop to stop you from stressing your eyes out or falling asleep.

## Different approaches to Image Recognition

The easiest way to get image recognition on your app is to use an online service. Today we are going to use a service called Azure, provided by Microsoft. Azure has an image classifier that has already been trained, and you can access it by subscribing to Computer Vision API.

For those of you who want to have a bit of flexibility in training the classifier, Azure also provides Custom Vision Services, where you just upload and tag images, and it automatically learns how to tag new images.

For ambitious programmers who already know how to use APIs, we will briefly cover how to create and train your own model using CNTK (Microsoft Cognitive Toolkit). It is a well-known machine learning framework comparable to TensorFlow, which would also be useful for building other machine learning models.

## For those of you who are new to programming

Today we will be using two programming languages, Python and JavaScript. Python is a handy language for processing information and data. JavaScript is a web-friendly language often used for interactive websites, and has evolved into a general-purpose language with the development of Node.js, which can run on the machine as well as in the browser.

Unlike more low-level (or computer-friendly) languages such as C++ and Java, Python and JavaScript has better human interpretability. We can use any language to use these Computer Vision APIs, but for the sake of simplicity we will be focusing on using Python and JavaScript today.

## Installing Python, Node.js and a code editor

The first thing you need to do is to download Python and Node.js – we need software packages that interpret these high-level programming languages and execute it on our machine. Download Python 3.6.4 and Node 8.9.4 and run the installer.

We run our code from a shell (command prompt / PowerShell for Windows, terminal for Mac). Some code editors can display the shell and the editing window at the same time which makes it very convenient and time efficient. In my demo, I'm going to use Visual Studio Code, which runs both on Windows, Mac and Linux. It is light and runs perfectly without any customisation. Alternatives are Atom and Sublime Text.

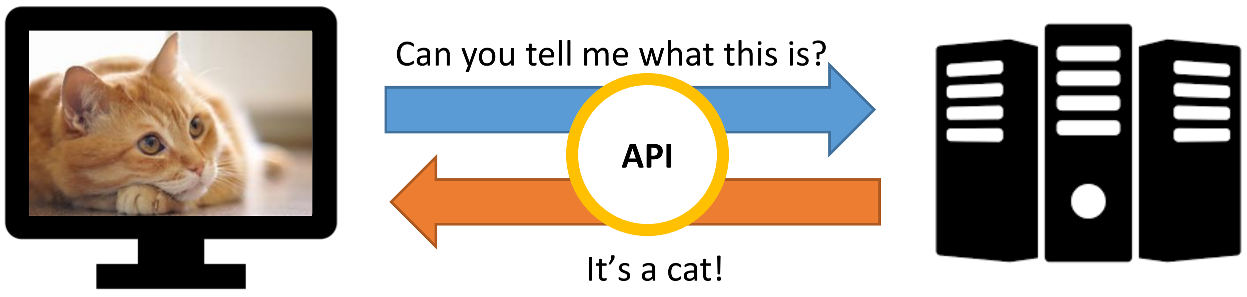
## What is an API?

Imagine you are working for a non-technical company which wants to use image recognition for its service. You have a week to develop a prototype. Would you want to build an image recogniser from scratch? Surely not!

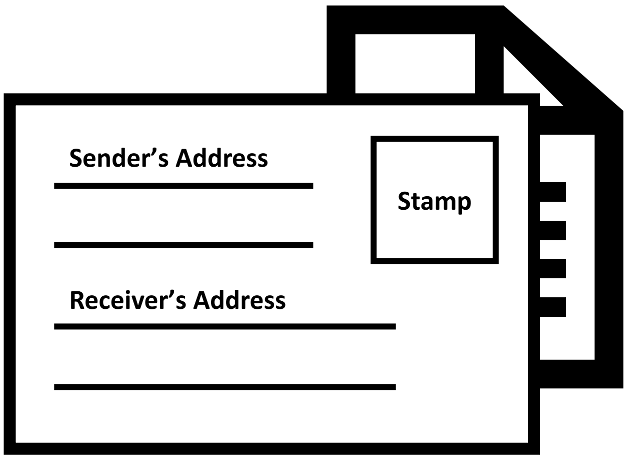
That is why big companies such as Microsoft, Google and IBM are providing online services that can already do this for you. They have computers running in their data centre (which are called ***servers***) which can do all the hard stuff for you. Once you are subscribed to the service, you just have to make a ***request*** to the ***server*** with whatever image data you want it to classify, and get a ***response***, telling you what the image is.

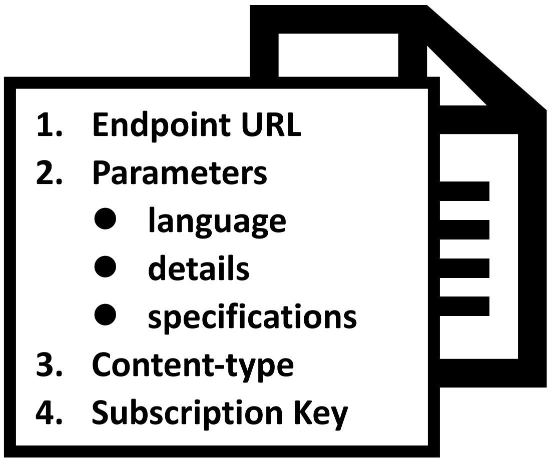
**API (Application Programming Interface)** is just about how the communication between your laptop and the server is defined. API is an interface between the laptop and the computer, similar to a remote controller that acts as an interface between the user and the TV. To use the service, you just have to press the right button.

So, if you send an image you want it to classify, you will get the information you require.



Now let's draw a slightly different analogy. Communication between a laptop and the server is like delivering a post, except that it is way faster. Like a post, the information sent have two parts – the ***header*** and the ***body***. The header is like the envelope, containing information of where the post is to be delivered. Instead of calling it a delivery address, we call this an ***endpoint URL***.





Often you pass details of the data you are passing, or specifications of the desired response as ***parameters***. For example, if you are passing a soundwave data to a speech recogniser, you want to tell it which language the speech is in, the sampling frequency of the soundwave, etc. and also specify what kind of response you want (whether you are just interested in getting the transcription, or if you also want alternatives, confidence, timestamps, etc.)

Unlike humans, who can read a letter without being told whether it is written in English, French or some other language, computers must be told which format the content is in. ***JSON*** is a popular format which is convenient for passing data (XML used to be another popular format). Therefore, we want to set the ***Content-type*** to **application/json**.

You want to include a ***subscription key*** that you will get upon enrolling to the API service. This is required since most companies don’t want to open up their service for charity. Once you start using APIs and cloud services, you will have tones of subscription keys for different services. Make sure you **never share your subscription keys online** on places like GitHub or YouTube.

Last but not least, you set the data you want to be processed in the **body** (the content). This will be the image you want to classify for this workshop, but it could be soundwaves, video URLs or other data depending on the API you are using. The body will be formatted in the format we have specified, in this case JSON.

# Getting started with Azure

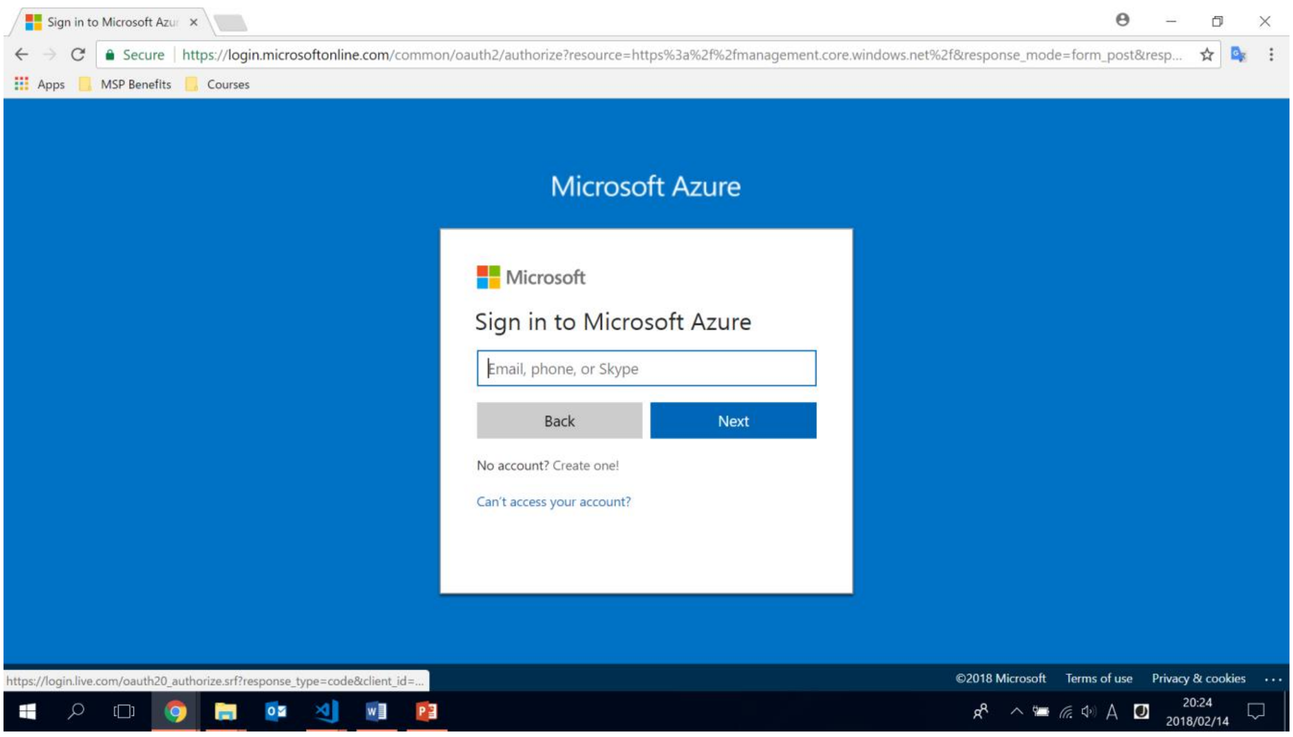
## What is Azure?

Many companies are providing cloud services nowadays. Cloud services can be providing storage, pre-configured environment where users can run applications care-free, or APIs that call out trained machine learning models.

Microsoft Azure is a major cloud service, comparable to Google Cloud Platform and AWS. Today we are going to use just a couple of its functionalities: Computer Vision API and Custom Vision API.

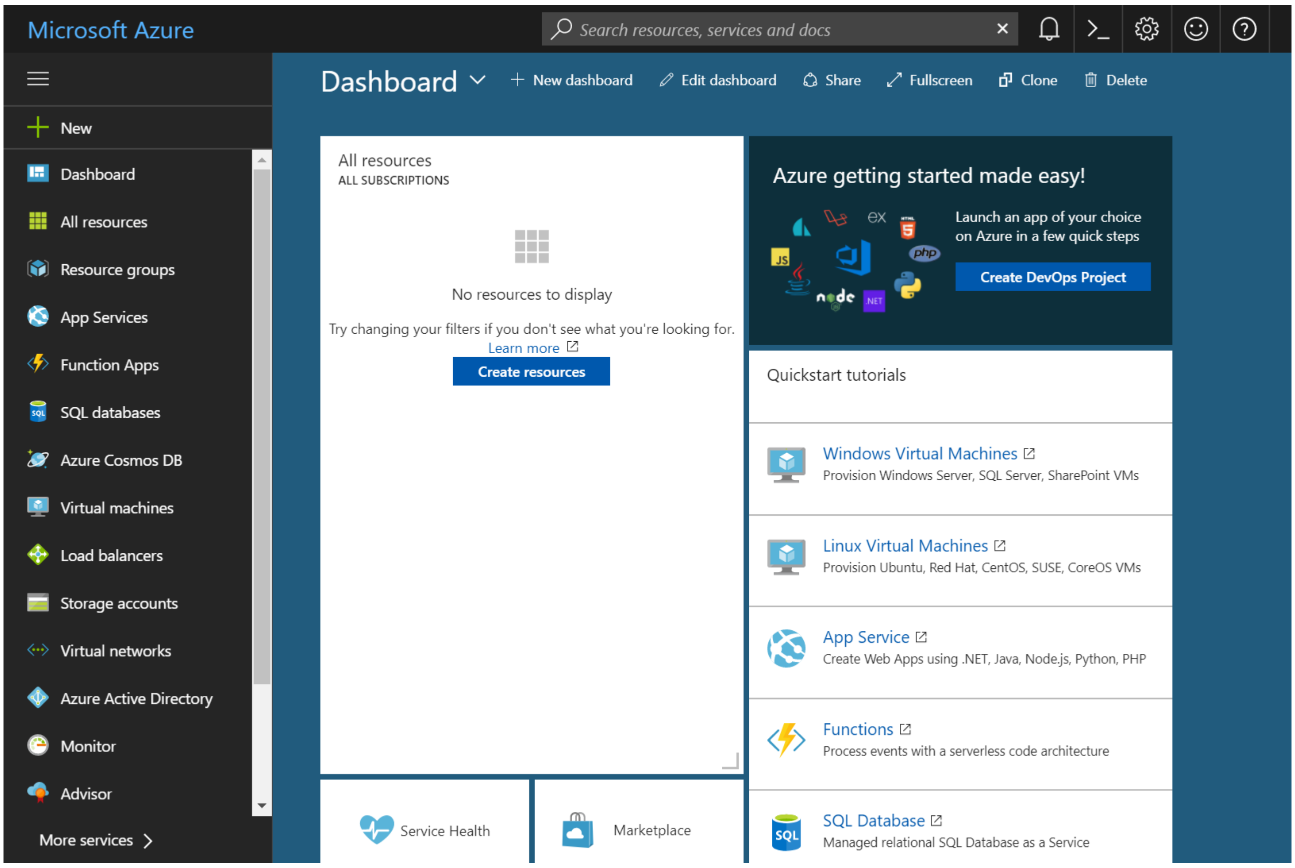
## Creating an Azure account

As for all services, we need an account to start using Azure. The steps are easy. Click on [https://portal.azure.com](https://portal.azure.com) which will direct you to the account creation page. If you want to know what you are signing up for, check out [https://azure.microsoft.com/free](https://azure.microsoft.com/free).


Sign in to Microsoft Azu
Secure
https://login.microsoftonline.com/common/oauth2/authorize?resource=https%3a%2f%2fmanagement.core.windows.net%2f&response_mode=form_post&resp...
Apps
MSP Benefits
Courses
Microsoft Azure
Microsoft
Sign in to Microsoft Azure
Email, phone, or Skype
Back
Next
No account? Create one!
Can't access your account?
https://login.live.com/oauth20_authorize.srf?response_type=code&client_id=...
©2018 Microsoft
Terms of use
Privacy & cookies
20:24
2018/02/14

Assuming you don't have an account yet, click **'Create one!'**. The next page will ask you to enter your email account details. If you want to open a Microsoft email account (outlook.com, hotmail.com, etc.) you can do so by clicking **'Get a new email address'**.

Once you are logged in, your Azure dashboard should show up.



All resources
ALL SUBSCRIPTIONS

No resources to display

Try changing your filters if you don't see what you're looking for.
Learn more

Azure getting started made easy!

Launch an app of your choice on Azure in a few quick steps

Quickstart tutorials

Windows Virtual Machines
Provision Windows Server, SQL Server, SharePoint VMs

Linux Virtual Machines
Provision Ubuntu, Red Hat, CentOS, SUSE, CoreOS VMs

App Service
Create Web Apps using .NET, Java, Node.js, Python, PHP

Functions
Process events with a serverless code architecture

SQL Database
Managed relational SQL Database as a Service

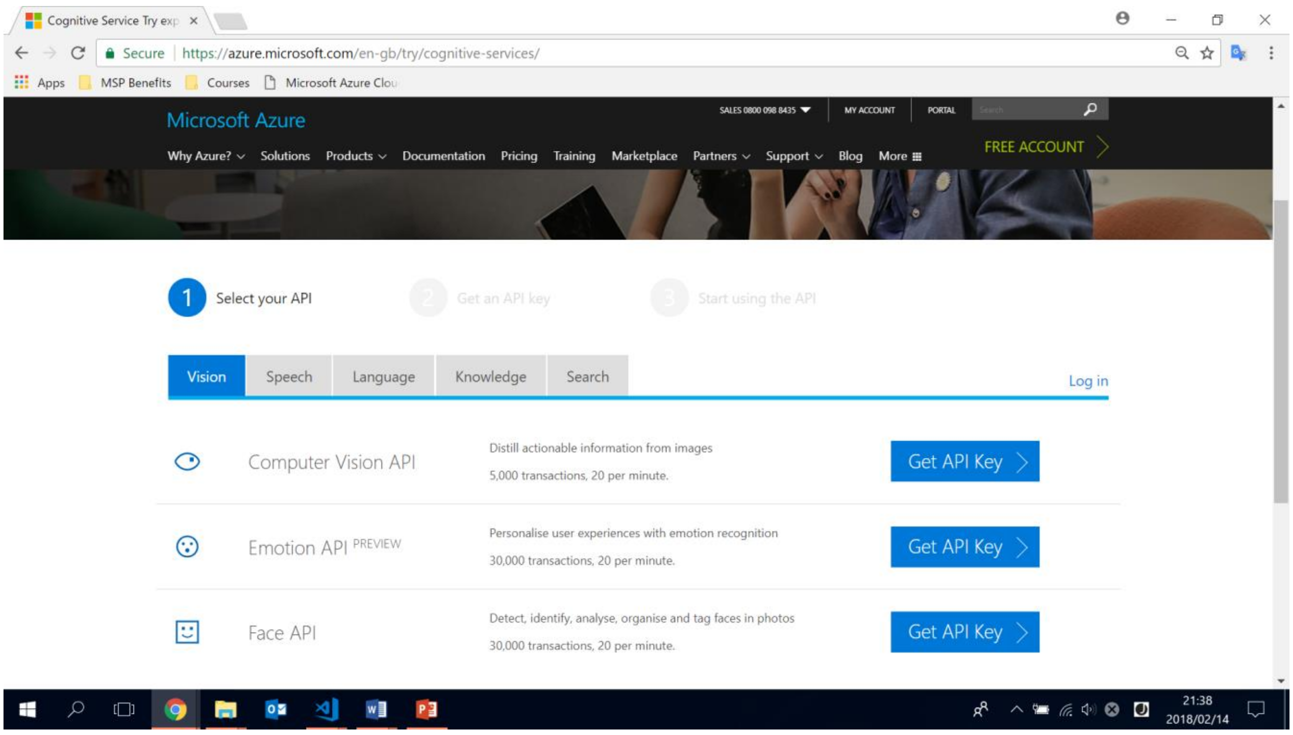
This is when you could claim your Azure Pass. Because you turned up to the workshop, you have won $100 worth of Azure Credit for free! The demonstrator will show you how to claim the promo code. Now, go to https://www.microsoftazurepass.com/. After a couple of clicks, you can enter the promo code and redeem your Azure Pass.

We wouldn't be exploring the full capacity of Azure – we will be focusing on APIs for today. However if you want to learn how you can use Azure to build your own machine learning model, come to the Machine Learning Workshop next Tuesday!

# Computer Vision API

## Subscribing to Computer Vision API

Computer Vision API is one of Azure Cognitive Services. Follow the link and click **'Try Cognitive Services for free'**, and you will find a button to get the API key.



| | | |
|---|---|---|
| Computer Vision API | Distill actionable information from images<br>5,000 transactions, 20 per minute. | Get API Key |
| Emotion API PREVIEW | Personalise user experiences with emotion recognition<br>30,000 transactions, 20 per minute. | Get API Key |
| Face API | Detect, identify, analyse, organise and tag faces in photos<br>30,000 transactions, 20 per minute. | Get API Key |

## Getting your first response

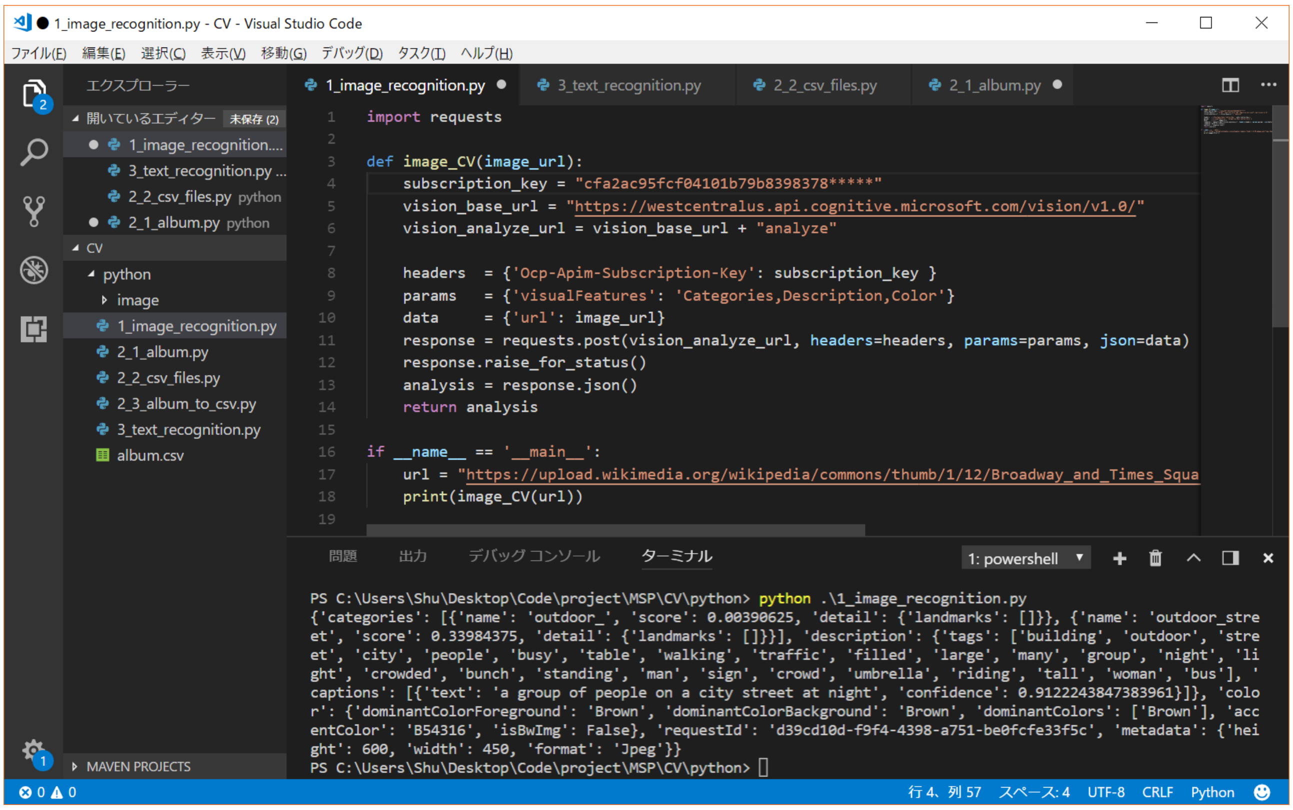
Have a look through the documentation. We will start with the Python documentation. The documentation walks you through the details of every line of the code. If you want to get to the code quickly, clone (or download) my GitHub repository and open **1_image_recognition.py**. Set the variable **'subscription_key'** to the API key for the Computer Vision API that you have obtained just now.

On the first line of the code, you will see "**import requests**". This is telling Python to import an additional library (a package of code for specific purposes), in this case a library called **requests**. If you are new to Python, you will probably have to download the library onto your computer. Fortunately, the process is simple. Just type **"pip install requests"** in your shell and it should automatically install. You should do the same for every new **import** statement if you get an error when you run the code.

Now, execute the code. Open the shell (if you are using Visual Studio Code, this should pop up at the bottom if you click on the **✖0⚠0** button) and change to the directory named python by typing the command **cd ./python**. Run the Python code by typing **python** ./**1_image_recognition.py**. You don't have to type in the entire file name. After typing the first couple of characters, press tab a couple of times to show options.

Did you see an output that looks like this?
{'categories': [{'name': 'outdoor_', 'score': 0.00390625, 'detail': {'landmarks': []}}, {'name': 'outdoor_street', 'score': 0.33984375, 'detail': {'landmarks': []}}], …

Great! Now you are properly communicating with the API!


1_image_recognition.py - CV - Visual Studio Code
ファイル(F) 編集(E) 選択(C) 表示(V) 移動(G) デバッグ(D) タスク(T) ヘルプ(H)
エクスプローラー
開いているエディター 未保存 (2)
1_image_recognition.py
3_text_recognition.py
2_2_csv_files.py
2_1_album.py
CV
python
image
2_3_album_to_csv.py
album.csv
MAVEN PROJECTS
問題 出力 デバッグ コンソール ターミナル
1: powershell
行 4、列 57 スペース: 4 UTF-8 CRLF Python

```python
import requests

def image_CV(image_url):
    subscription_key = "cfa2ac95fcf04101b79b8398378*****"
    vision_base_url = "https://westcentralus.api.cognitive.microsoft.com/vision/v1.0/"
    vision_analyze_url = vision_base_url + "analyze"

    headers  = {'Ocp-Apim-Subscription-Key': subscription_key }
    params   = {'visualFeatures': 'Categories,Description,Color'}
    data     = {'url': image_url}
    response = requests.post(vision_analyze_url, headers=headers, params=params, json=data)
    response.raise_for_status()
    analysis = response.json()
    return analysis

if __name__ == '__main__':
    url = "https://upload.wikimedia.org/wikipedia/commons/thumb/1/12/Broadway_and_Times_Squa
    print(image_CV(url))
```

```
PS C:\Users\Shu\Desktop\Code\project\MSP\CV\python> python .\1_image_recognition.py
{'categories': [{'name': 'outdoor_', 'score': 0.00390625, 'detail': {'landmarks': []}}, {'name': 'outdoor_stre
et', 'score': 0.33984375, 'detail': {'landmarks': []}}], 'description': {'tags': ['building', 'outdoor', 'stre
et', 'city', 'people', 'busy', 'table', 'walking', 'traffic', 'filled', 'large', 'many', 'group', 'night', 'li
ght', 'crowded', 'bunch', 'standing', 'man', 'sign', 'crowd', 'umbrella', 'riding', 'tall', 'woman', 'bus'], '
captions': [{'text': 'a group of people on a city street at night', 'confidence': 0.9122243847383961}]}, 'colo
r': {'dominantColorForeground': 'Brown', 'dominantColorBackground': 'Brown', 'dominantColors': ['Brown'], 'acc
entColor': 'B54316', 'isBwImg': False}, 'requestId': 'd39cd10d-f9f4-4398-a751-be0fcfe33f5c', 'metadata': {'hei
ght': 600, 'width': 450, 'format': 'Jpeg'}}
PS C:\Users\Shu\Desktop\Code\project\MSP\CV\python>
```

Now let's go through each step. The function **image_CV(image_url)** receives the URL to the image data you want to analyse and returns the response from the API in **JSON** format. In the function, we are using the library called **requests**, which POSTs the request to the server as I have explained in the introduction. It does most of the technical stuff for you so you can focus on just passing the necessary parameters. We are passing the **vision_analyze_url** as the **endpoint**, the **subscription_key**, which visual features we want to extract as **parameters**, and the image URL.

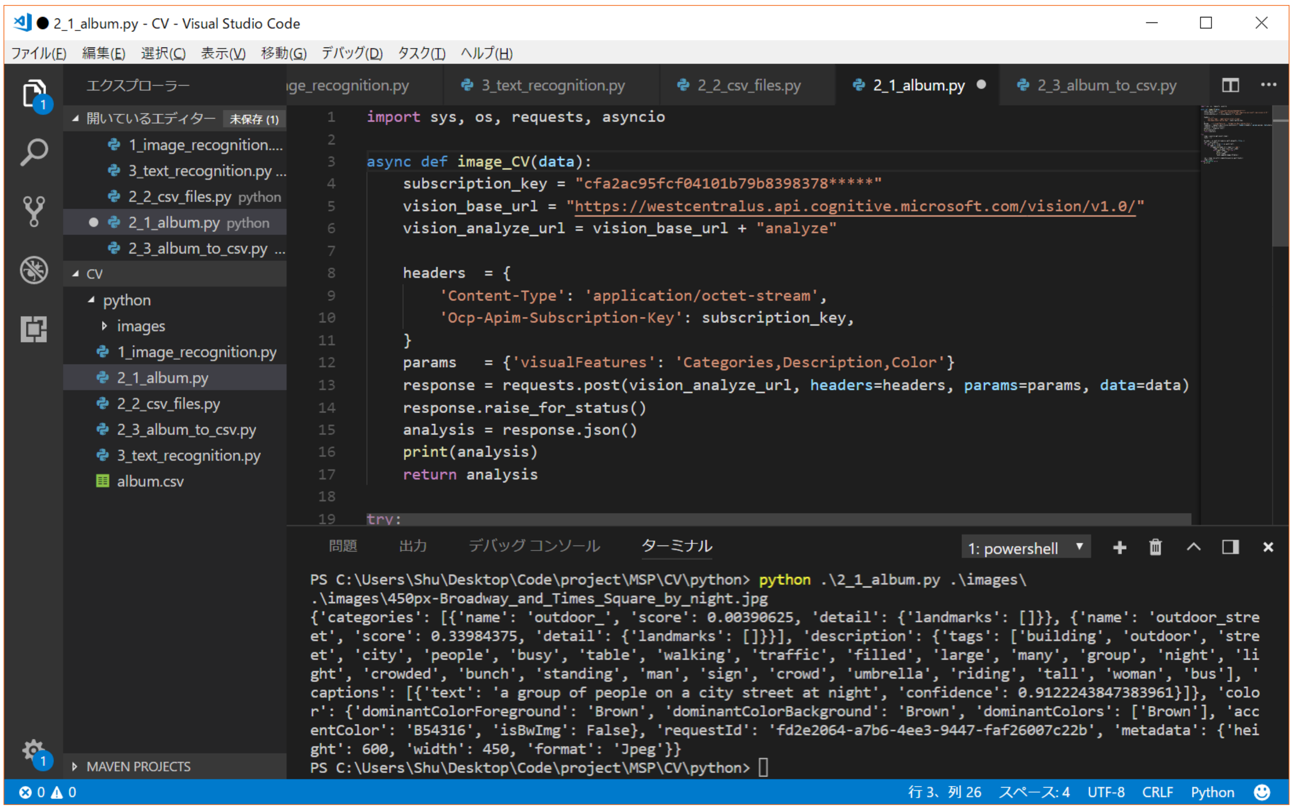
JSON is a useful format since the information is well structured (either in a list or a dictionary format) and each element could be accessed from calling the **key**. For instance, if I have a JSON object as below, I can call out the values by:
me = {'name': 'Shu', 'age': 21, 'course': 'engineering'}
my_name = me['name']

## Sort out your cluttered photo collection

Image recognition is great if you don't want to spend hours on labelling them or searching for the image you want. In this section, we'll see how to navigate through locally stored images (photos on your laptop) and label them automatically.

The challenge here is the delay due to the time the API takes to analyse every image. We want to loop through all the images contained in a directory and subdirectory and send those images to the API repeatedly. It would take a long time before we finish labelling all of them if we have to wait for the API to return each response before sending another request. This is why we should use **asynchronous** function calls. Instead of waiting for each operation to complete, we create a list of tasks and run those tasks as they come along.

Check out **2_1_album.py**. Now the script loops through files in the directory that you will specify, and calls out **async def image_CV(data)** whenever it finds an image. Note that now the function is receiving the image data rather than the image URL. The data is in computer readable binary format. According to the documentation, "the binary image data must be passed in via the **data parameter** to **requests.post** as opposed to the **json parameter**." You also have to indicate that you are passing binary data as your content of the request, so set **'Content-Type': 'application/octet-stream'**.



```python
import sys, os, requests, asyncio

async def image_CV(data):
    subscription_key = "cfa2ac95fcf04101b79b8398378*****"
    vision_base_url = "https://westcentralus.api.cognitive.microsoft.com/vision/v1.0/"
    vision_analyze_url = vision_base_url + "analyze"

    headers  = {
        'Content-Type': 'application/octet-stream',
        'Ocp-Apim-Subscription-Key': subscription_key,
    }
    params   = {'visualFeatures': 'Categories,Description,Color'}
    response = requests.post(vision_analyze_url, headers=headers, params=params, data=data)
    response.raise_for_status()
    analysis = response.json()
    print(analysis)
    return analysis

try:
```

```
PS C:\Users\Shu\Desktop\Code\project\MSP\CV\python> python .\2_1_album.py .\images\
.\images\450px-Broadway_and_Times_Square_by_night.jpg
{'categories': [{'name': 'outdoor_', 'score': 0.00390625, 'detail': {'landmarks': []}}, {'name': 'outdoor_stre
et', 'score': 0.33984375, 'detail': {'landmarks': []}}], 'description': {'tags': ['building', 'outdoor', 'stre
et', 'city', 'people', 'busy', 'table', 'walking', 'traffic', 'filled', 'large', 'many', 'group', 'night', 'li
ght', 'crowded', 'bunch', 'standing', 'man', 'sign', 'crowd', 'umbrella', 'riding', 'tall', 'woman', 'bus'], '
captions': [{'text': 'a group of people on a city street at night', 'confidence': 0.9122243847383961}]}, 'colo
r': {'dominantColorForeground': 'Brown', 'dominantColorBackground': 'Brown', 'dominantColors': ['Brown'], 'acc
entColor': 'B54316', 'isBwImg': False}, 'requestId': 'fd2e2064-a7b6-4ee3-9447-faf26007c22b', 'metadata': {'hei
ght': 600, 'width': 450, 'format': 'Jpeg'}}
PS C:\Users\Shu\Desktop\Code\project\MSP\CV\python> 
```

Run the python code. Now you have to pass an additional parameter to indicate which folder contains the images you want analysed. I called this folder **images**, so I can run **“python .¥2_1_album.py .¥images¥”**. Feel free to add more images into this folder to get more responses. It might take a while if the image is large.

## Export your results to a CSV file

Okay, seeing the results on the shell is great, but maybe not very useful for practical purposes. Let’s briefly talk about saving the results in a CSV file.

Comma-Separated Values (CSV) file is a useful and light-weight representation of table structure data. You can open this file on Excel and it is widely compatible to other spreadsheet applications. You will see CSVs used in machine learning and other information analytics as well.

**2_2_csv_files.py** shows you how to create a CSV file. Once you have run the code and are satisfied, open **2_3_album_to_csv.py** which basically exports your results that you got in **2_1_album.py** to a CSV file named ‘album.csv’. It extracts the caption and tags and stores them along with the image file path.

## Text recognition – make your hand-written notes permanent!

Probably the most useful part of the technology for us students is text recognition. Amazingly, Computer Vision API not only transcribe typed text but also recognises hand-written text, even with cursive writing! (given that it’s also legible for humans…)

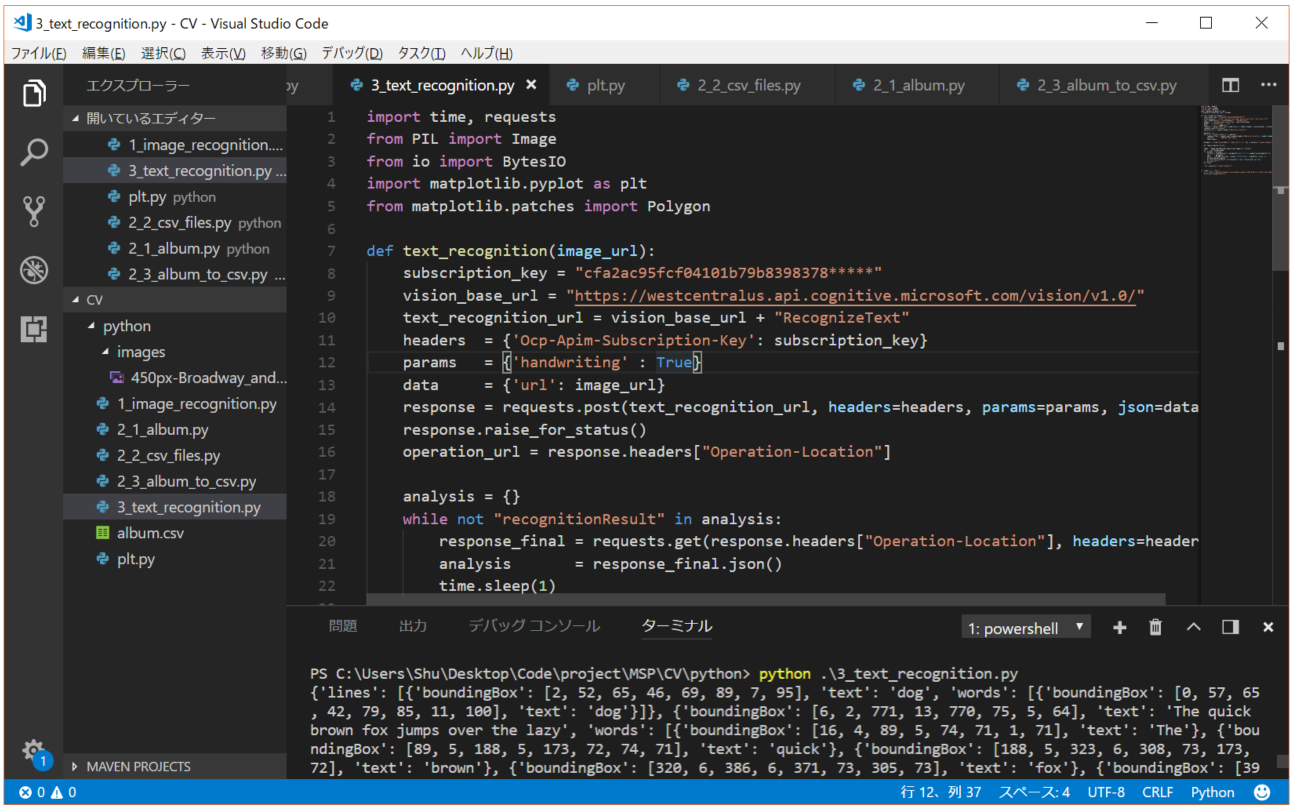
This part of the documentation talks about how to use the text recognition capability of the API. The tricky part is that, unlike the previous implementations, the text recognition service does not return the recognized text by itself. Instead, it returns an "Operation Location" URL. We have to call this URL separately once the recognition result is ready.

Run **3_text_recognition.py**. The result will look like this:

```
{'lines': [{'boundingBox': [2, 52, 65, 46, 69, 89, 7, 95], 'text': 'dog', 'words': [{'boundingBox': [0, 57, 65, 42, 79, 85, 11, 100], 'text': 'dog'}]}, {'boundingBox': [6, 2, 771, 13, 770, 75, 5, 64], 'text': 'The quick brown fox jumps over the lazy', 'words': [{'boundingBox': [16, 4, 89, 5, 74, 71, 1, 71], 'text': 'The'}, …
```

You will notice that there are a lot of information given about the location of the text. This would be useful for reconstructing the entire collection of texts onto the 2D image.

Here, we are using **matplotlib** to overlay the text onto the original image. Matplotlib is another famous library for showing different kind of graphs and images.



```python
import time, requests
from PIL import Image
from io import BytesIO
import matplotlib.pyplot as plt
from matplotlib.patches import Polygon

def text_recognition(image_url):
    subscription_key = "cfa2ac95fcf04101b79b8398378*****"
    vision_base_url = "https://westcentralus.api.cognitive.microsoft.com/vision/v1.0/"
    text_recognition_url = vision_base_url + "RecognizeText"
    headers  = {'Ocp-Apim-Subscription-Key': subscription_key}
    params   = {'handwriting' : True}
    data     = {'url': image_url}
    response = requests.post(text_recognition_url, headers=headers, params=params, json=data
    response.raise_for_status()
    operation_url = response.headers["Operation-Location"]

    analysis = {}
    while not "recognitionResult" in analysis:
        response_final = requests.get(response.headers["Operation-Location"], headers=header
        analysis       = response_final.json()
        time.sleep(1)
```

```
PS C:\Users\Shu\Desktop\Code\project\MSP\CV\python> python .\3_text_recognition.py
{'lines': [{'boundingBox': [2, 52, 65, 46, 69, 89, 7, 95], 'text': 'dog', 'words': [{'boundingBox': [0, 57, 65
, 42, 79, 85, 11, 100], 'text': 'dog'}]}, {'boundingBox': [6, 2, 771, 13, 770, 75, 5, 64], 'text': 'The quick
brown fox jumps over the lazy', 'words': [{'boundingBox': [16, 4, 89, 5, 74, 71, 1, 71], 'text': 'The'}, {'bou
ndingBox': [89, 5, 188, 5, 173, 72, 74, 71], 'text': 'quick'}, {'boundingBox': [188, 5, 323, 6, 308, 73, 173,
72], 'text': 'brown'}, {'boundingBox': [320, 6, 386, 6, 371, 73, 305, 73], 'text': 'fox'}, {'boundingBox': [39
```

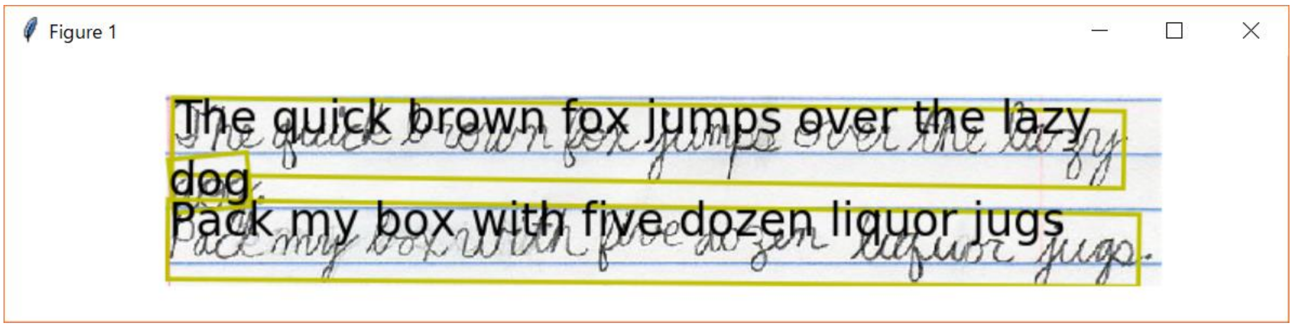


Figure 1
The quick brown fox jumps over the lazy
dog
Pack my box with five dozen liquor jugs

## Mini-challenge: transcribe your lecture notes

For experienced programmers, now is the time for a hack! Similar to what we have done for the **album_to_csv** programme, write a programme that transcribes all the pictures of your lecture notes / white boards, and export it into text files. Maybe you can name the text file according to the date and time of the lecture, or you can even extract the title of the slide from the picture.

If that is not enough, you can even look into indenting and styling the text according to the position and sizes of each of the boxes containing the block of text.

# Custom Vision Services

## When to choose Custom Vision Services

So far, we have explored different features of Computer Vision API, which has been trained to suit high-demand purposes such as labelling objects and extracting text. There are other features to it as well that we haven't explored, such as identifying celebrities and landmarks.

However, sometimes this general-purpose API isn't enough to serve one's needs. Maybe you want to build a classifier that is trained to identify subtle features, or to return a very detailed classification (a tag 'plant' is not detailed enough if you want to classify different species of plants). That is when Custom Vision Services will be useful.

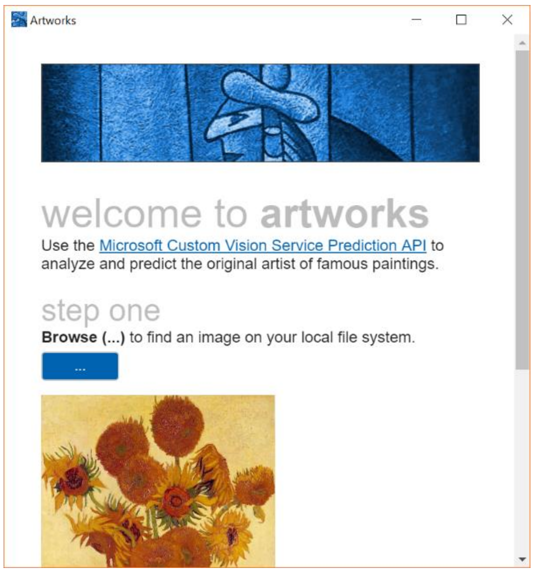
Custom Vision Services is by far the easiest way to train your own classifier. You just have to upload a dozen of images and label them according to how you want it to classify your image. That's really it! You immediately have a trained model that can go above 90% precision and recall, which you can visually examine.

## Getting Started with Custom Vision Services

I would walk through you the steps, but there is already an easy-to-follow tutorial prepared by Microsoft. Cloning the GitHub repository is a slow process since everything is under one massive repository, so I've already included the necessary files under my repository. Once you've followed each step of the tutorial, you should get an app running as shown on the right.

In fact, this tutorial is also a great introduction to **Node.js**, which is server-side JavaScript to run web applications. What is more, you can create a desktop application with an extension called **Electron**, which we are using for this tutorial. No more hacking around GUI and window objects in C# - this alternative gives you a very quick-to-build solution that enables you to develop desktop apps just using HTML/CSS.

Now you are set to create your own image classifier on Custom Vision Services.



welcome to artworks

Use the Microsoft Custom Vision Service Prediction API to analyze and predict the original artist of famous paintings.

step one

**Browse (...)** to find an image on your local file system.

# Treasure Hunt on Street View

## Project example using Computer Vision and Custom Vision

Now let me show you a working example of how you can combine Computer Vision and Custom Vision with other services e.g. Google Maps and Street View to create an entertaining game. In this project, we will run a treasure hunting app that detects objects within Google Street View using image recognition.

## Objective of the treasure hunt

We have a treasure box we want to open, but it requires many keys in different colours to open. These keys could be found in Google Street View. You are given a list of objects / landscapes to look for in the Street View within a given time limit. Once you find an item of interest in the Street View, press the "Analyse Image" button, which will then send the view to the Microsoft Computer Vision API. If you find the correct item, you acquire a key!

**[ YouTube Video: https://youtu.be/zVgNC38ijfk ]**

## Where to find my project

Check out my hackathon submission at https://devpost.com/software/streasure-hunt where you would be able to find my demo video, project description and an online working demo.

All the code is available on the GitHub repository under "Streasure Hunt". Examine the code and insert your own API key. Then enter "**node app.js"** into the terminal, and go to http://127.0.0.1:3000/ to see the app running.

## Getting started - creating your treasure map

The first stage of the project is to import Google Street View and Google Maps (or *treasure map*, as I like to think of it) and linking them up. This documentation shows how to show Google Maps and Google Street View side-by-side. Navigating around either of them will automatically refresh the view shown on the other, which is very convenient for our treasure hunt.

Another thing we have to do is to capture the image shown on the Street View whenever the user presses the "Analyse Image" button. Conveniently, Google provides an API to retrieve a street view as an image by posting information of the position and orientation you desire. Go to the Google Street View Image API documentation and write a function that retrieves the thumbnail image URL whenever the user presses the "Analyse Image" button. You will need position and orientation parameters such as POV, heading, pitch and FOV. How to obtain these are shown here. The tricky bit is to obtain FOV.

```
function getViewUrl(){
    var position = panorama.getPosition()+'';
    position = position.replace(/[()]/g, "").replace(" ","");
    var heading = panorama.getPov().heading;
    var pitch = panorama.getPov().pitch;
    var zoom = panorama.getZoom();
    var fov = 180 / Math.pow(2,zoom);
    var imageUrl =
"https://maps.googleapis.com/maps/api/streetview?size=640x640&location="
+position+"&heading="+heading+"&pitch="+pitch+"&fov="+fov+"&key="+MAP_AP
I_KEY;
    return imageUrl;
}
```

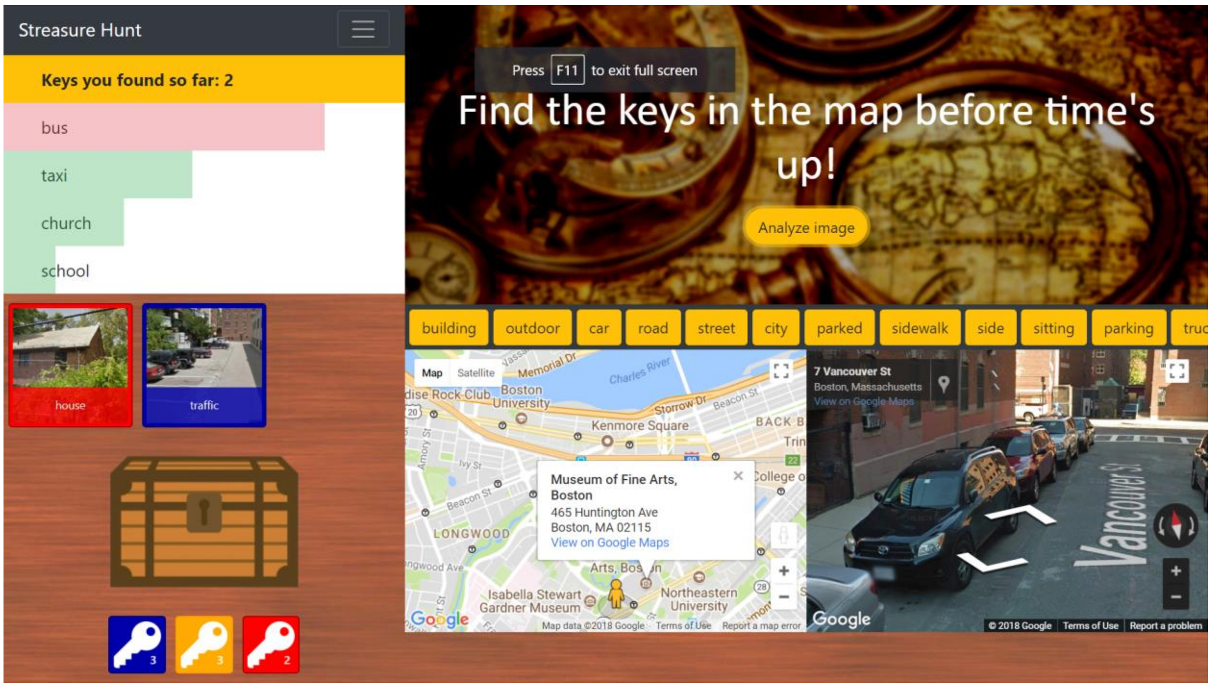
Go to https://console.developers.google.com/apis/library and enable **Google Street View Image API** and **Google Maps JavaScript API**. Then activate your API key and insert it into your code where it says ***YOUR_API_KEY***. Now we have everything we need from Street View.

## Microsoft Azure Computer Vision API

Now let's analyse Street View images using Microsoft Computer Vision API!

Here is the documentation of the API using JavaScript. The use of the API is simple – just insert your own API key where it says "subscription key", and parse the JSON response. You can get the API key here. For our scenario, we just need the "tags" under "description".

I created a visual representation of the tags and compared them with the item list on the left. If there is a match, the item is added to the key gallery below. Creating the item list was the major part of the work. Check out how to make a to-do list at w3schools.



## Microsoft Custom Vision Service

Now let's think about assigning colours to the keys you acquire so that you have to play clever when you find the keys. Here, I chose a specific property that could be used because of the nature of how we acquire the data. All images are taken from Google Street View, which means that no matter where in the world you want to play this game, the image will always include either a road or a roadside view. The blue keys mean the road extends forwards, the yellow ones mean you have the road at an angle, and the red ones mean you are looking into the roadside.

Go to https://www.customvision.ai/ and create a project. Near the top of your workspace, there is a button named "Add images". For this game, I uploaded around 100 Google Street View images with tags of "forwards", "angles" and "sides".

Click on the green button on the top. On the performance tab, we can see how well the prediction is by looking at the precision and recall of the model. We could try improving the model by feeding in more image data.

Once the model is trained, click the "Prediction URL" in the "performance" page. The API key to request predictions will be shown. Although there isn't a written example of using the API with JavaScript in the documentation, how you make a request to the API is basically the same as the Computer Vision API (or most APIs in fact).

# Classification with Keras

## If you want to have full control over your model

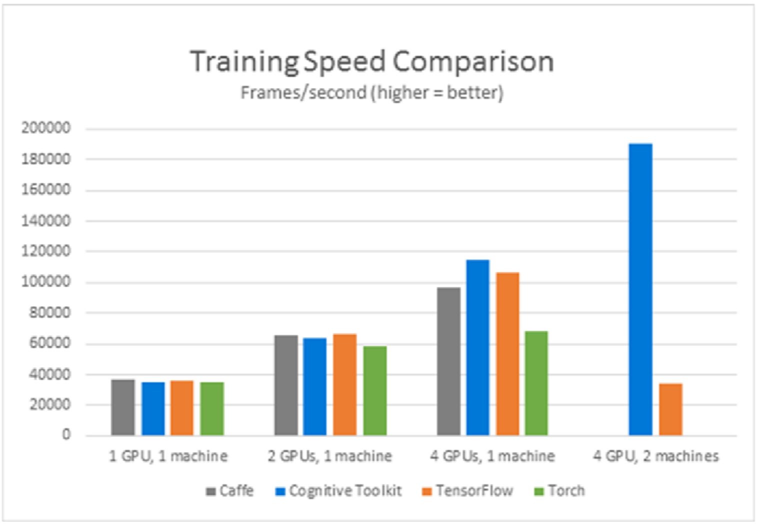
So far, we have covered how to use Microsoft API services to classify our images. This method has a couple of advantages: you only have to write a minimum amount of code; you don't have to train your own machine learning model with terabytes of image data, which could take up both memory and time; it has higher accuracy than the model you can train; usually the API response is fast enough compared to models run locally on your machine.

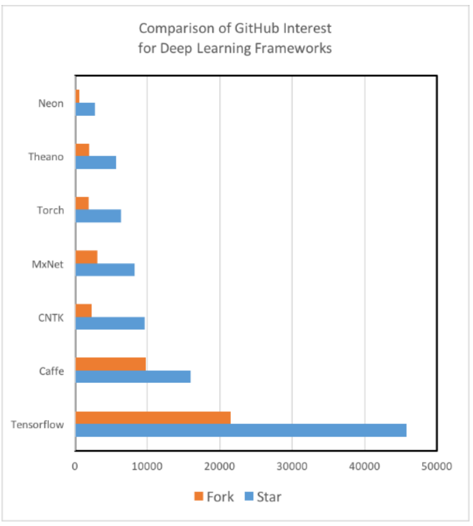
However, sometimes you want to have the fullest control over your model, changing the number or size of layers in your Neural Networks, or detecting latent features. Or maybe you just want to have a good enough classifier for a product without having to pay for the API subscription.

This is when you want to start building your own Neural Network model. Thankfully, we don't have to implement Neural Networks from zero. There are many libraries for machine learning and Neural Networks available which we could use.

## Machine Learning made easy by frameworks

Many machine learning frameworks have been developed by large companies. Caffe, CNTK, TensorFlow and Torch are some of the popular frameworks available. Cognitive Toolkit (CNTK) developed by Microsoft outperforms other frameworks for multiple GPUs and is the go-for framework for distributed computing, while TensorFlow developed by Google has strong community support and resources.





Using these frameworks has significant advantages:

1. You don't have to know about the mathematical details of Neural Networks, which involves a lot of calculus and linear algebra. You can just determine the structure of the Neural Network you want to build, and the framework will take care of implementing the algorithm.
2. Better computation speed could be achieved by using frameworks, since it is compiled and run on a machine friendly format. They typically convert a series of vector and matrix operations into what is called **computation graphs**, which is then compiled and run on C++ or some other low-level language. Frameworks like CNTK and TensorFlow also take advantage of GPUs. This helps increasing the training speed of the model.
3. Because you are not writing basic low-level functions but can focus on the high-level implementations, you can quickly test your ideas and achieve a faster development loop.

## Installing CNTK / TensorFlow

How you want to install CNTK / TensorFlow onto your machine depends on the OS and the environment you are working on, but generally it could be achieved by a simple pip installation. How to install CNTK / How to install TensorFlow

## Introduction to Keras

CNTK and TensorFlow are great, but is still a complicated framework for beginners. To make things easier, we can use a front-end framework called Keras; it is a user-oriented framework that helps you build Neural Networks with minimum coding. It works together with CNTK, TensorFlow or Theano (another ML framework) as a backend, and runs models using its capability.

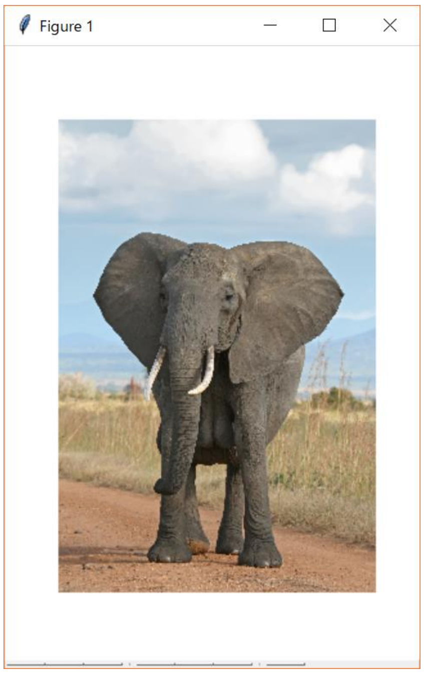
Run pip install to install Keras.

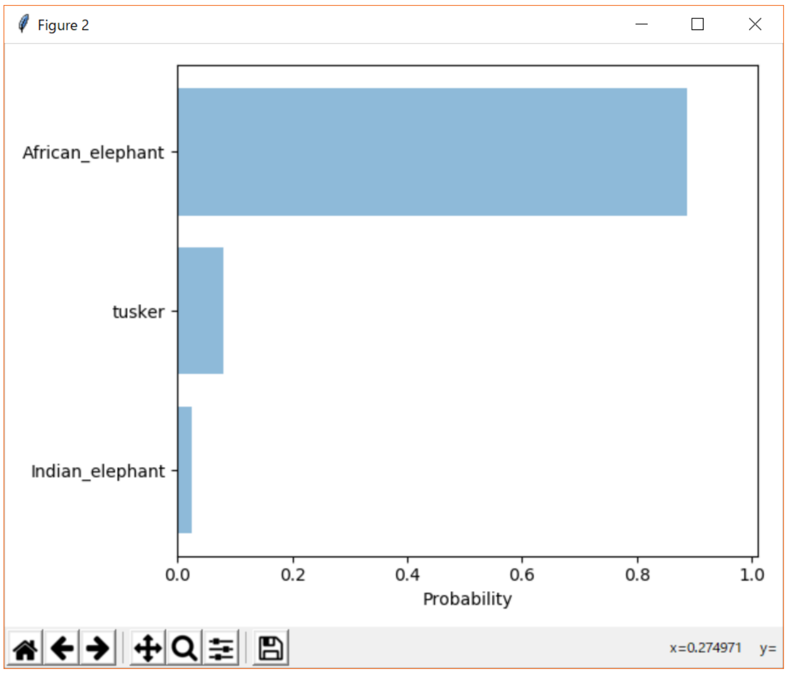
The great thing about Keras is that you can test out the same Neural Network on different backends by simply modifying one line in the configuration. See this documentation to learn how to swap your backend.

## Image recognition with Keras

I would like to give credit to Greg Chu for this blog article on how to build an image classifier using Keras. Go to the GitHub repository and clone it onto your machine. The chapter for image recognition essentially addresses the problem we solved by using Microsoft Computer Vision API earlier in the workshop. This Neural Network downloads the parameters already trained with ImageNet dataset (a large dataset of labelled images used for general computer vision training and validation purposes) so that it gives us the predicted labels without us having to train the model ourselves.

Similar to the Computer Vision API, this code allows us to feed the image data either as binary data or by passing the image URL.





The second chapter discusses the concept of transfer learning. Rather than training an image classification model from scratch for every single custom model we build, which could take up weeks to train before it can identify useful features to classify objects, it is way more efficient and reliable to use pre-trained model parameters for the first couple of layers for your Neural Network, and then to append some custom trainable layers at the very end to train the model specific for your purpose.

This is the secret behind why Microsoft Custom Vision Services achieve such high prediction accuracy with just 6 or 7 training images in a short training time. It already has pre-trained Neural Network layers and is customising the network according to your training dataset using the last few layers.

# Finally

## Prototyping an idea

We have covered different methods and uses of image recognition. If you are developing an app, it is great to know that you can mount computer vision onto your app in 5 min, just by using APIs or frameworks. This will make it easy for you to build a prototype, enabling you to test and improve your idea rapidly.

## About Microsoft Student Partners

Microsoft Student Partners (MSP) is a programme run by Microsoft to encourage students to share knowledge and passion for technology on campus. Microsoft provides students access to their technology and resources, and generously sponsors us to run workshops, speaker events and hackathons. I would like to acknowledge Microsoft for providing us free access to their cloud platform Azure with $100 extra credit, and offering pizza, drinks to host this event.

## Future events

We will be organising more tech events later this term and the next academic year. Please follow our Facebook page and subscribe to our mailing list for more information.

Furthermore, we are recruiting for new members to join our Microsoft Student Partner committee for the next academic year. If you are a 1st year / 2nd year student in any discipline who is interested in running these events, please reach out to us.

**Thank you very much for taking part in the workshop!**