# Image recognition - Identify rooms of the house using Neural Network

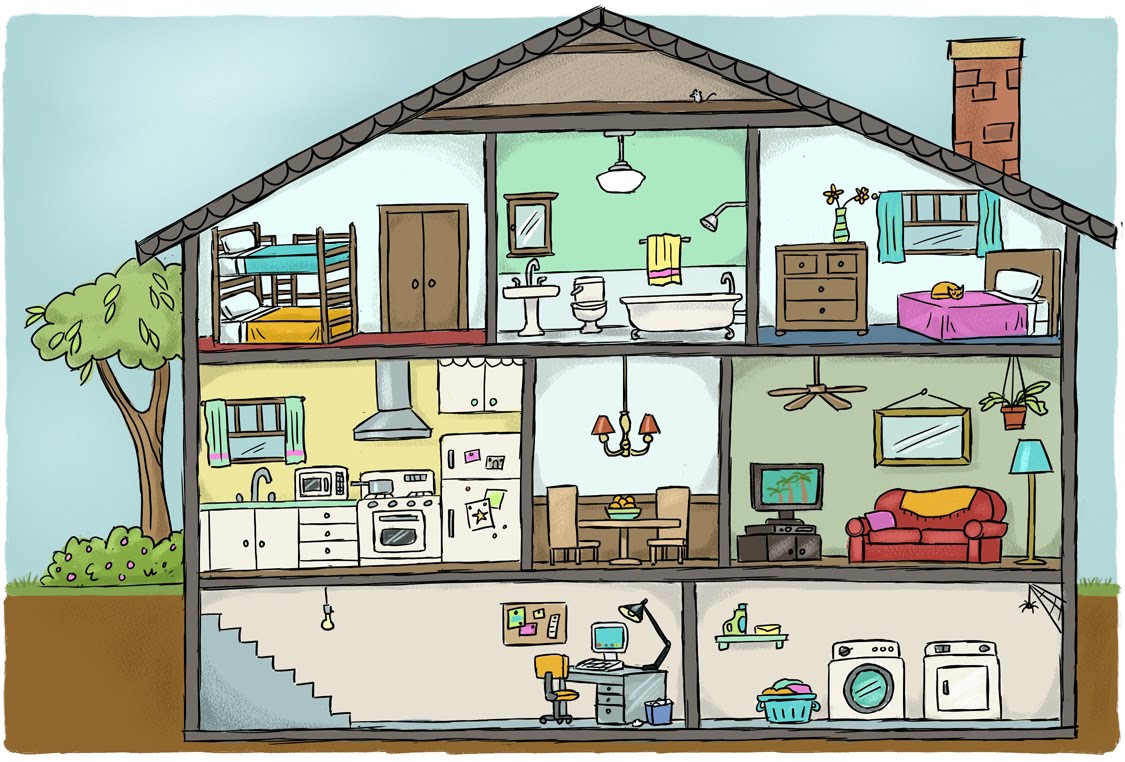


Image recognition, in the context of machine vision, is the ability of software to [identify objects](https://whatis.techtarget.com/definition/object-recognition), places, people, writing and actions in images. Computers can use machine vision technologies in combination with a camera and [artificial intelligence](https://searchenterpriseai.techtarget.com/definition/AI-Artificial-Intelligence) software to achieve image recognition.

Image recognition is used to perform a large number of machine-based visual tasks, such as labeling the content of images with [meta-tags](https://whatis.techtarget.com/definition/meta-description-tag), performing image content search and guiding autonomous robots, self-driving cars and accident avoidance systems.

While human and animal brains recognize objects with ease, computers have difficulty with the task. Software for image recognition requires [deep machine learning](https://searchenterpriseai.techtarget.com/definition/deep-learning-deep-neural-network). Performance is best on convolutional [neural net processors](https://searchenterpriseai.techtarget.com/definition/neural-network) as the specific task otherwise requires massive amounts of power for its compute-intensive nature. Image recognition [algorithms](https://whatis.techtarget.com/definition/algorithm) can function by use of comparative [3D models](https://whatis.techtarget.com/definition/3D-model), appearances from different angles using edge detection or by components. Image recognition algorithms are often trained on millions of pre-labeled pictures with guided computer learning.

Current and future applications of image recognition include smart photo libraries, targeted advertising, the interactivity of media, accessibility for the visually impaired and enhanced research capabilities. [Google](https://searchcio.techtarget.com/definition/Google-The-Company), [Facebook](https://whatis.techtarget.com/definition/Facebook), [Microsoft](https://searchwindowsserver.techtarget.com/definition/Microsoft), [Apple](https://whatis.techtarget.com/definition/Apple) and [Pinterest](https://whatis.techtarget.com/definition/Pinterest) are among the many companies that are investing significant resources and research into image recognition and related applications. Privacy concerns over image recognition and similar technologies are controversial as these companies can pull a large volume of data from user photos uploaded to their social media platforms.

In the analysis below neural networks and deep learning algorithms will be used to identify which room of a household the image is depicting.

One use case there the final algorithm can be used in is for smart vacuum cleaners, where they will have to perform different cleaning processes depending on the room of the house they are at. In general there could potentially be numerous application with the rise of IoT.

# Data collection – Web scraping



To **gather the images google image** search engine was scraped for specific search queries like ‘kitchen’, ‘old kitchen’, ‘modern kitchen’, ‘bathroom’ etc

More specifically the steps taken to gather our labeled images were the following:

* Logging output for after code runs

logging.basicConfig(filename='myapp.log', level=logging.INFO)

agent\_version = '%.2f' % (random.randint(20, 100) + random.randint(1, 100)/float(100))

* Create a header for the get request when downloading the images

headers = {

'User-Agent': 'Mozilla/5.0 (X11; Linux x86\_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/56.0.2924.87 Safari/537.36',

'User-Agent': 'Mozilla/5.0 (compatible; MSIE 8\.0; Windows NT 5\.1; SV1) Chrome/%s.2924.87 Safari/537.36' % agent\_version}

os = 'linux'

* Open a browser but not visible

display = Display(visible=0, size=(800, 800))

display.start()display = Display(visible=0, size=(800, 800))

display.start()

* Google url for image searches

basic\_url = 'https://www.google.gr/search?q={}&source=lnms&tbm=isch&sa=X&ved=0ahUKEwjT-Kj668vbAhUD\_iwKHSlVAKwQ\_AUICigB&biw=1280&bih=615'

* For every search term include search term in google url

url = basic\_url.format(str(param))

* Run chrome driver for selenium to start the scraping process

driver = webdriver.Chrome("/usr/lib/chromium-browser/chromedriver")

* Give time for selenium to start

wait = WebDriverWait(driver, 100)

* Perform get request for the google url

driver.get(str(url))

* Sometimes google opens a bot page first wait until class name 'med'. It waits up to 100 seconds

wait.until(lambda driver: driver.find\_elements(By.CLASS\_NAME, 'med'))

* Scroll down to reveal more pictures

for scroll in range(10):

driver.execute\_script("window.scrollBy(0,1000000)")

time.sleep(0.2)

time.sleep(0.5)

* List to store the image urls after scrapping the google results

img\_urls = []

* Find all photos, this is where in the html from google we can find the images

images = driver.find\_elements\_by\_xpath("//div[@class='rg\_meta notranslate']")

print('---- Number of pictures found: ', len(images))

for image in images:

get image information from every html code chunk. the information i am retrieving is the image url

img\_url = json.loads(image.get\_attribute("innerHTML"))["ou"]

* Create an id for the image url we retrieved, this is the name i am going to give to the image

img\_id = str(uuid.uuid4())

* Store image url and image name in a dictionary

img\_dict = {'img\_url':img\_url, 'img\_id':img\_id}

* Append the above information for every loop i.e. for every image

img\_urls.append(img\_dict)

* Close selenium

We only use selenium to get the information from google image search. once we have the image urls we just do a simple get request. No need for selenium driver.

driver.quit()

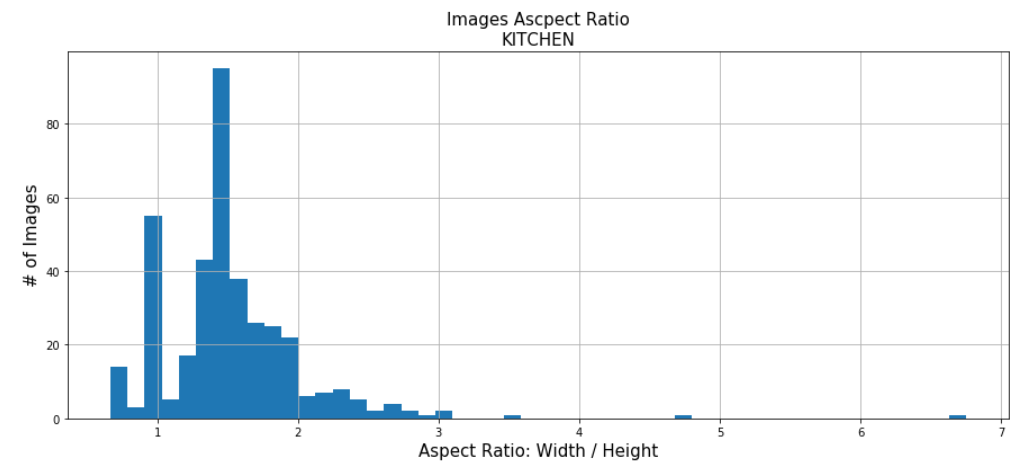
* Create dir if it does not exist
* For every search term create the respective path file in order to store the
* Download the pictures, for every image url

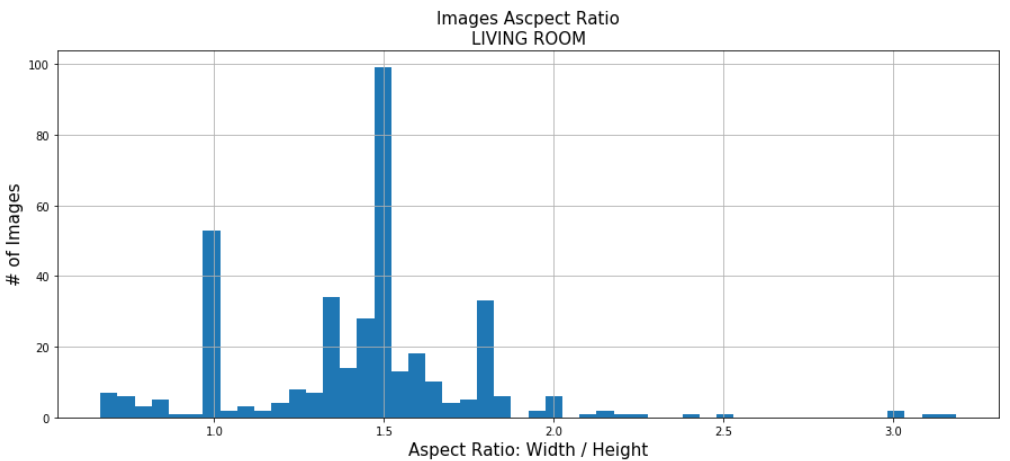
# Data Processing

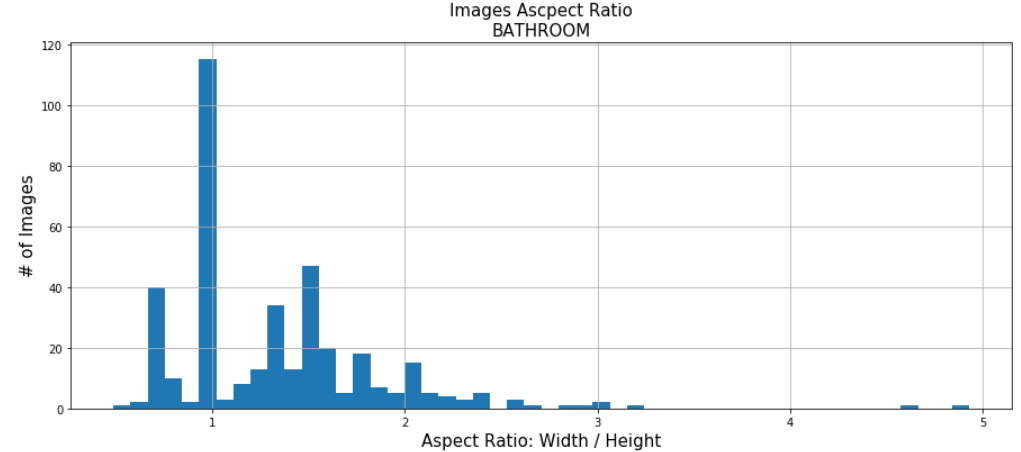
The biggest challenge in terms of data wrangling is to **resize** the images so as all image to be the same, in terms of size and **reshape** them in a form that can be processed.

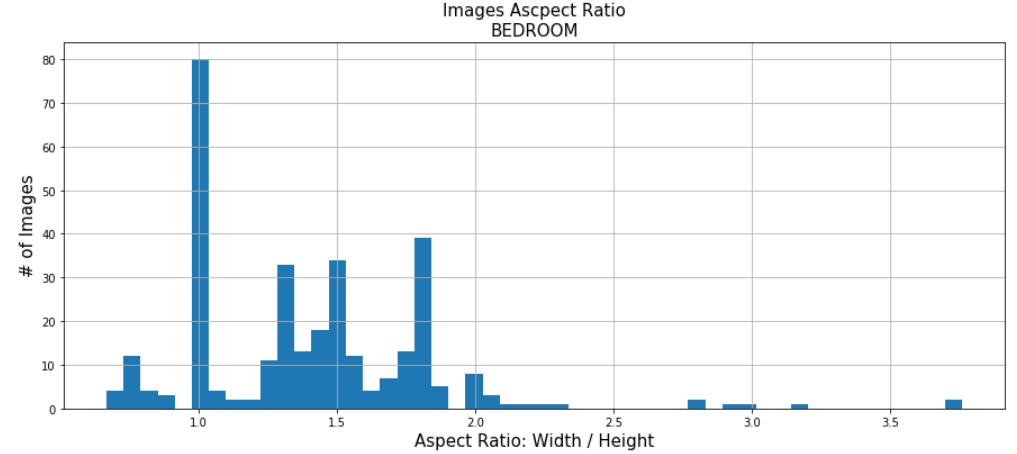
Resizing:

As we can see from the charts above the size of the pictures varies a lot. In order to proceed with the analysis of the images have to have the same size. So all photos have been resized to 100x100.









Before the resizing we need to get image sizes and paths

sizes = defaultdict(list)

imgs = defaultdict(list)

for path in paths:

pics = os.listdir(path)

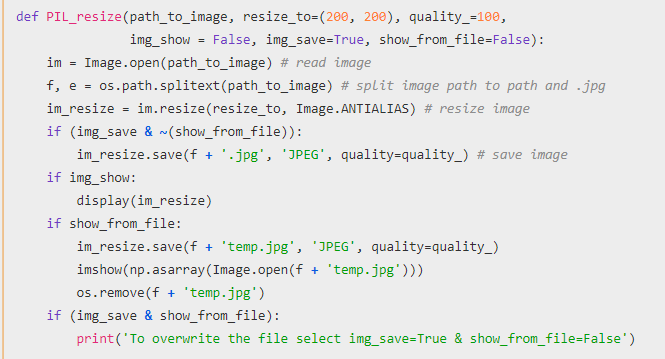
for pic in pics:

try:

sizes[path.split('/')[-2]].append(Image.open(path+pic).size)

imgs[path.split('/')[-2]].append(path+pic)

We have created a custom function to resize the images:



Then the resizing prosses can follow. The pictures below are an example of a picture before and after the resizing.

imshow(np.asarray(Image.open(imgs['livingroom'][10])));



PIL\_resize(path\_to\_image=imgs['livingroom'][120], resize\_to=(100,100), quality\_=100, img\_show=False, img\_save=False, show\_from\_file=True)



Reshaping

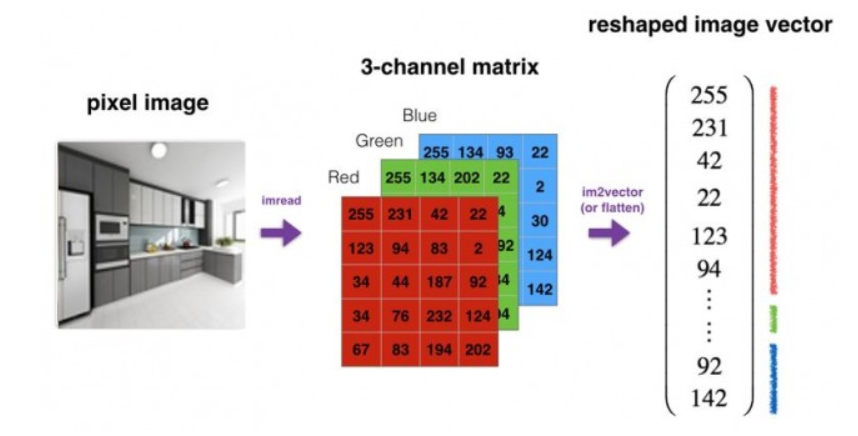
Image to vector conversion: Since we have chosen the 100x100 size we have 30000 pixels / RGB per image.

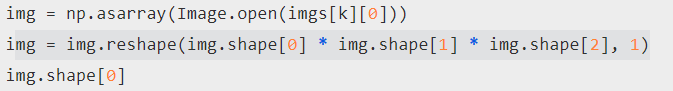
img = np.asarray(Image.open(imgs[k][0]))

img = img.reshape(img.shape[0] \* img.shape[1] \* img.shape[2], 1)

img.shape[0]

Reshaping:





We have 3000 pixels / RGB per image

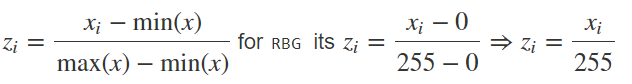
We create an initial matrix X of dimantions n, m where n is the number of pixels \* rgb colours and m are the number of images

X = np.zeros((img.shape[0], imgs\_count))

The labeled data of the image are stored in list Y

Y = list()

For every image we have stored we create respective X and Y we open the images convert the image to a vector of size x\_pixels \* y\_pixels \* 3, 1 and store it in the respective column of matrix X, for example the first image in going to be stored in the first column of X. We normalize using the nim-mx method.



X[:, i] = img.reshape(img.shape[0] \* img.shape[1] \* img.shape[2]) / 255

We then append the folder name in which the image was saved.