Image Classification basics

mylearningsinaiml.wordpress.com/deep-learning/dl4cv/image-classification-basics/

May 28, 2018

"A picture is worth a thousand words" – English idiom

It simply means that a complex idea can be conveyed in a single image. However, for computers, interpreting the contents of an image is less trivial – all our computer sees is a big matrix of numbers. It has no idea regarding the thoughts, knowledge, or meaning the image is trying to convey.

In order to understand the contents of an image, we must apply **image classification**, which is the task of using computer vision and machine learning algorithms to extract meaning from an image.

Ex : It could be as simple as assigning a label to what the image contains, or as advanced as interpreting the contents of an image and returning a human-readable sentence.

Image classification and **image understanding** are currently the most popular sub-field of computer vision. We'll see more and more consumer applications on our smartphones that can ***understand and interpret the contents of an image***.

What Is Image Classification?

Image classification, at its very core, is the task of *assigning a label to an image from a predefined set of categories*.

–> analyze an input image and return a label that categorizes the image. The label is always from a predefined set of possible categories.

Our classification system could also assign multiple labels to the image via probabilities.

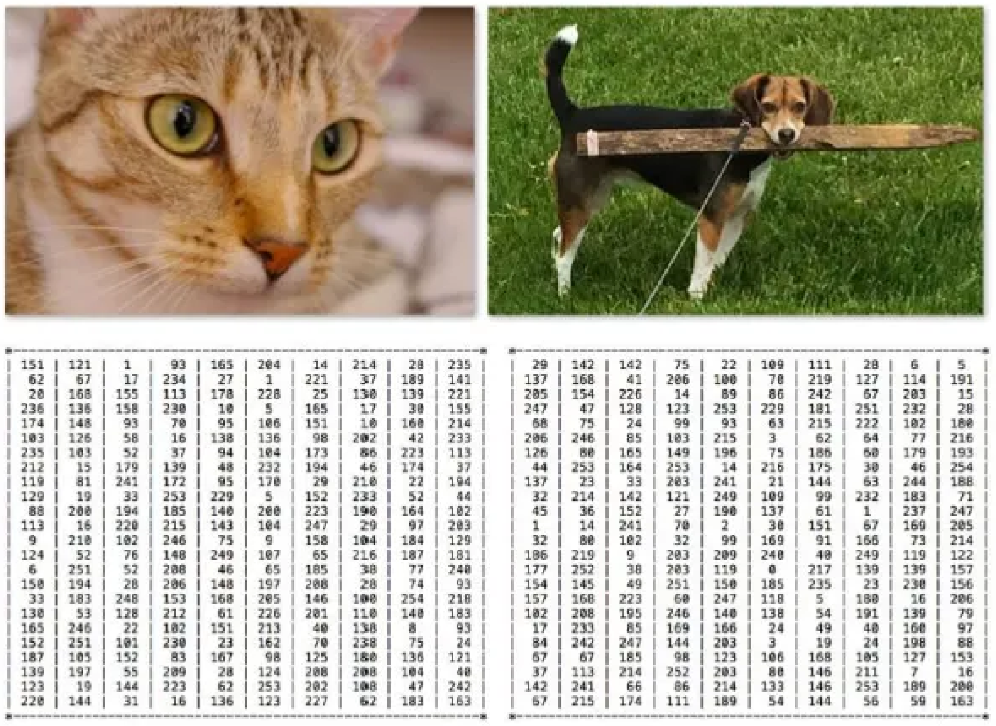
More formally, given our input image of W x H pixels with three channels, Red, Green, and Blue, respectively, our goal is to take the W x H x 3=N pixel image and figure out how to correctly classify the contents of the image.

A **dataset** is therefore a collection of **data points.**

Our goal is to apply a machine learning and deep learning algorithms to discover underlying patterns in the dataset, enabling us to correctly classify data points that our algorithm has not encountered yet.

Semantic Gap : Human Vs Computer

Take a look at the two photos below. It should be fairly trivial for us to tell the difference between the two photos – there is clearly a cat on the left and a dog on the right. But all a computer sees is two big matrices of pixels (bottom).



```
| 151 | 121 |   1 |  93 | 165 | 204 |  14 | 214 |  28 | 235 |
|  62 |  67 |  17 | 234 |  27 |   1 | 221 |  37 | 189 | 141 |
|  20 | 168 | 155 | 113 | 178 | 228 |  25 | 130 | 139 | 221 |
| 236 | 136 | 158 | 230 |  10 |   5 | 165 |  17 |  30 | 155 |
| 174 | 148 |  93 |  70 |  95 | 106 | 151 |  10 | 160 | 214 |
| 103 | 126 |  58 |  16 | 138 | 136 |  98 | 202 |  42 | 233 |
| 235 | 103 |  52 |  37 |  94 | 104 | 173 |  86 | 223 | 113 |
| 212 |  15 | 179 | 139 |  48 | 232 | 194 |  46 | 174 |  37 |
| 119 |  81 | 241 | 172 |  95 | 170 |  29 | 210 |  22 | 194 |
| 129 |  19 |  33 | 253 | 229 |   5 | 152 | 233 |  52 |  44 |
|  88 | 200 | 194 | 185 | 140 | 200 | 223 | 190 | 164 | 102 |
| 113 |  16 | 220 | 215 | 143 | 104 | 247 |  29 |  97 | 203 |
|   9 | 210 | 102 | 246 |  75 |   9 | 158 | 104 | 184 | 129 |
| 124 |  52 |  76 | 148 | 249 | 107 |  65 | 216 | 187 | 181 |
|   6 | 251 |  52 | 208 |  46 |  65 | 185 |  38 |  77 | 240 |
| 150 | 194 |  28 | 206 | 148 | 197 | 208 |  28 |  74 |  93 |
|  33 | 183 | 248 | 153 | 168 | 205 | 146 | 100 | 254 | 218 |
| 130 |  53 | 128 | 212 |  61 | 226 | 201 | 110 | 140 | 183 |
| 165 | 246 |  22 | 102 | 151 | 213 |  40 | 138 |   8 |  93 |
| 152 | 251 | 101 | 230 |  23 | 162 |  70 | 238 |  75 |  24 |
| 187 | 105 | 152 |  83 | 167 |  98 | 125 | 180 | 136 | 121 |
| 139 | 197 |  55 | 209 |  28 | 124 | 208 | 208 | 104 |  40 |
| 123 |  19 | 144 | 223 |  62 | 253 | 202 | 108 |  47 | 242 |
| 220 | 144 |  31 |  16 | 136 | 123 | 227 |  62 | 183 | 163 |
```

```
|  29 | 142 | 142 |  75 |  22 | 109 | 111 |  28 |   6 |   5 |
| 137 | 168 |  41 | 206 | 100 |  70 | 219 | 127 | 114 | 191 |
| 205 | 154 | 226 |  14 |  89 |  86 | 242 |  67 | 203 |  15 |
| 247 |  47 | 128 | 123 | 253 | 229 | 181 | 251 | 232 |  28 |
|  68 |  75 |  24 |  99 |  93 |  63 | 215 | 222 | 102 | 180 |
| 206 | 246 |  85 | 103 | 215 |   3 |  62 |  64 |  77 | 216 |
| 126 |  80 | 165 | 149 | 196 |  75 | 186 |  60 | 179 | 193 |
|  44 | 253 | 164 | 253 |  14 | 216 | 175 |  30 |  46 | 254 |
| 137 |  23 |  33 | 203 | 241 |  21 | 144 |  63 | 244 | 188 |
|  32 | 214 | 142 | 121 | 249 | 109 |  99 | 232 | 183 |  71 |
|  45 |  36 | 152 |  27 | 190 | 137 |  61 |   1 | 237 | 247 |
|   1 |  14 | 241 |  70 |   2 |  30 | 151 |  67 | 160 | 205 |
|  32 |  80 | 102 |  32 |  99 | 169 |  91 | 166 |  73 | 214 |
| 186 | 219 |   9 | 203 | 209 | 240 |  40 | 249 | 119 | 122 |
| 177 | 252 |  38 | 203 | 119 |   0 | 217 | 139 | 139 | 157 |
| 154 | 145 |  49 | 251 | 150 | 185 | 235 |  23 | 230 | 156 |
| 157 | 168 | 223 |  60 | 247 | 118 |   5 | 180 |  16 | 206 |
| 102 | 208 | 195 | 246 | 140 | 138 |  54 | 191 | 139 |  79 |
|  17 | 233 |  85 | 169 | 166 |  24 |  49 |  40 | 160 |  97 |
|  84 | 242 | 247 | 144 | 203 |   3 |  19 |  24 | 198 |  88 |
|  67 |  67 | 185 |  98 | 123 | 106 | 168 | 105 | 127 | 153 |
|  37 | 113 | 214 | 252 | 203 |  80 | 146 | 211 |   7 |  16 |
| 142 | 241 |  66 |  86 | 214 | 133 | 146 | 253 | 189 | 200 |
|  67 | 215 | 174 | 111 | 189 |  54 | 144 |  56 |  59 | 163 |
```

Figure 4.3: **Top:** Our brains can clearly see the difference between an image that contains a *cat* and an image that contains a *dog*. **Bottom:** However, all a computer "sees" is a big matrix of numbers. The difference between how we perceive an image and how the image is represented (a matrix of numbers) is called the *semantic gap*.

The semantic gap is the difference between how a human perceives the contents of an image versus how an image can be represented in a way a computer can understand the process.

Again, a quick visual examination of the two photos above can reveal the difference between the two species of an animal. But in reality, the computer has no idea there are animals in the image to begin with.

How do we go about encoding all this information in a way that a computer can understand it? The answer is to apply ***feature extraction*** to quantify the contents of an image. Feature extraction is the process of taking an input image, applying an algorithm, and obtaining a feature vector (i.e., a list of numbers) that quantifies our image.

We can apply deep learning to *automatically learn a set of features* that can be used to quantify and ultimately label the contents of the image itself.

However, it's not that simple. . . because once we start examining images in the real world, we are faced with many, many challenges.

What are the Challenges ??

If the semantic gap were not enough of a problem, we also have to handle factors of **variation** in how an image or object appears.

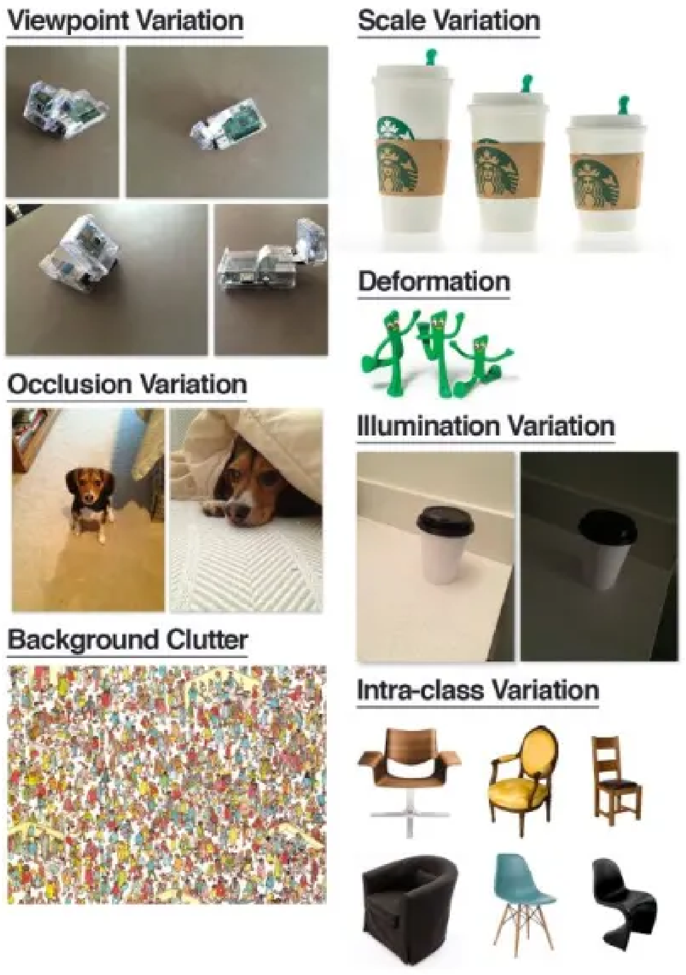


Figure 4.5: When developing an image classification system, we need to be cognizant of how an object can appear at varying viewpoints, lighting conditions, occlusions, scale, etc.

To start, we have **viewpoint variation**, where an object can be oriented/rotated in multiple dimensions with respect to how the object is photographed and captured.

We also have to account for **scale variation** as well. An object will look dramatically different when it is photographed up close versus when it is captured from farther away. Our image classification methods must be tolerable to these types of scale variations.

One of the hardest variations to account for is **deformation**. All images contain the an object; however, they are all dramatically different from each other.

Our image classification should also be able to handle **occlusions**, where large parts of the object we want to classify are hidden from view in the image.

Just as challenging as the deformations and occlusions mentioned above, we also need to handle the changes in **illumination**. (lighting conditions under which the image is captured)

Also, we have **intra-class variation**. The canonical example of intra-class variation in computer vision is displaying the diversification of chairs. From comfy chairs that we use to curl up and read a book, to chairs that line our kitchen table for family gatherings, to ultra-modern art deco chairs found in prestigious homes, a chair is still a chair – and our image classification algorithms must be able to categorize all these variations correctly.

Our image classification systems must also handle multiple variations combined together!

Successful computer vision, image classification, and deep learning systems deployed to the real-world make *careful assumptions and considerations* before a single line of code is ever written.

Always ***consider the scope of your image classifier***. If you take too broad of an approach, such as “I want to classify and detect every single object in my kitchen”, (where there could be hundreds of possible objects) then your classification system is unlikely to perform well unless you have years of experience building image classifiers – and even then, there is no guarantee to the success of the project.

But if you frame your problem and make it narrow in scope, such as “I want to recognize just stoves and refrigerators”, then your system is much more likely to be accurate and functioning, especially if this is your first time working with image classification and deep learning.

While deep learning and Convolutional Neural Networks have demonstrated significant robustness and classification power under a variety of challenges, you still should keep the scope of your project as tight and well-defined as possible.

** Keep in mind that **ImageNet**, the de-facto standard benchmark dataset for image classification algorithms, consists of 1,000 objects that we encounter in our everyday lives – and this dataset is still actively used by researchers trying to push the state-of-the-art for deep learning forward.

Semi-supervised learning is especially useful in computer vision where it is often time consuming, tedious, and expensive (at least in terms of man-hours) to label each and every single image in our training set. In cases where we simply do not have the time or resources to label each individual image, we can label only a tiny fraction of our data and utilize semi-supervised learning to label and classify the rest of the images.

The Deep Learning Classification Pipeline

Instead of writing a rule-based system, we instead take a *data driven approach* by supplying examples of what each category looks like and then teach our algorithm to recognize the difference between the categories using these examples.

We call these examples our training dataset of labeled images.

Step #1: Gather Your Dataset

The first component of building a deep learning network is to **gather our initial dataset**. We need the images themselves as well as the labels associated with each image. These labels should come from a finite set of categories, such as: categories = dog, cat, panda.

Furthermore, the number of images for each category should be approximately uniform (i.e., the same number of examples per category). If we have twice the number of cat images than dog images, and five times the number of panda images than cat images, then our classifier will become naturally biased to over-fitting into these heavily-represented categories.

Class imbalance is a common problem in machine learning and there exist a number of ways to overcome it.

Step #2: Split Your Dataset

Now that we have our initial dataset, we need to split it into two parts :

1. A training set
2. A testing set

A **training set** is used by our classifier to “learn” what each category looks like by making predictions on the input data and then correct itself when predictions are wrong. After the classifier has been trained, we can evaluate the performing on a **testing set.**

** It’s extremely important that the training set and testing set are independent of each other and do not overlap! You must keep this testing set entirely separate from your training process and use it only to evaluate your network.

These data splits make sense, **but what if you have parameters to tune**? Neural networks have a number of knobs and levers (ex., learning rate, decay, regularization, etc.) that need to be tuned and dialed to obtain optimal performance. We’ll call these types of parameters hyper-parameters, and it’s critical that they get set properly.

In practice, we need to test a bunch of these hyper-parameters and identify the set of parameters that works the best. You might be tempted to use your testing data to tweak these values, but again, this is a major no-no! The test set is only used in evaluating the performance of your network.

Instead, you should create a third data split called the **validation set**. This set of the data (normally) comes from the training data and is used as "fake test data" so we can tune our hyper-parameters. Only after have we determined the hyper-parameter values using the validation set do we move on to collecting final accuracy results in the testing data.

We normally allocate roughly 10-20% of the training data for validation.

Step #3: Train Your Network

Given our training set of images, we can now train our network. The goal here is for our network to learn how to recognize each of the categories in our labeled data. When the model makes a mistake, it learns from this mistake and improves itself.

So, how does the actual "learning" work? In general, we apply a form of **gradient descent.**

Step #4: Evaluate

Last, we need to evaluate our trained network. For each of the images in our testing set, we present them to the network and ask it to predict what it thinks the label of the image is. We then tabulate the predictions of the model for an image in the testing set.

Finally, these model predictions are compared to the ground-truth labels from our testing set. The ground-truth labels represent what the image category actually is. From there, we can compute the number of predictions our classifier got correct and compute aggregate reports such as precision, recall, and f-measure, which are used to quantify the performance of our network as a whole.

Feature-based Learning versus Deep Learning for Image Classification

In the traditional, feature-based approach to image classification, there is actually a step inserted between Step #2 and Step #3 – this step is *feature extraction*. During this phase, we apply hand engineered algorithms to quantify the contents of an image based on a particular component of the image we want to encode (i.e., shape, color, texture). Given these features, we then proceed to train our classifier and evaluate it.

When building Convolutional Neural Networks(CNNs), we can actually skip the feature extraction step. The reason for this is because CNNs are end-to-end models. We present the raw input data (pixels) to the network. The network then learns filters inside its hidden layers that can be used to discriminate amongst object classes. The output of the network is then a probability distribution over class labels.

*** One of the exciting aspects of using CNNs is that we no longer need to fuss over hand engineered features – we can let our network learn the features instead.

What Happens When my Predictions Are Incorrect?

Inevitably, you will train a deep learning network on your training set, evaluate it on your test set (finding that it obtains high accuracy), and then apply it to images that are outside both your training and testing set – only to find that the network performs poorly.

This problem is called **generalization**, the ability for a network to generalize and correctly predict the class label of an image that does not exist as part of its training or testing data.

The ability for a network to generalize is quite literally the most important aspect of deep learning research. If we can train networks that can generalize to outside datasets without retraining or fine-tuning, we'll make great strides in machine learning, enabling networks to be re-used in a variety of domains.

Instead of becoming frustrated with your model not correctly classifying an image, *consider the set of factors of variation* mentioned above. Does your training dataset accurately reflect examples of these factors of variation? If not, you'll need to gather more training data.