Handling Images and Text

Introduction to Data Science

Kigali, Rwanda

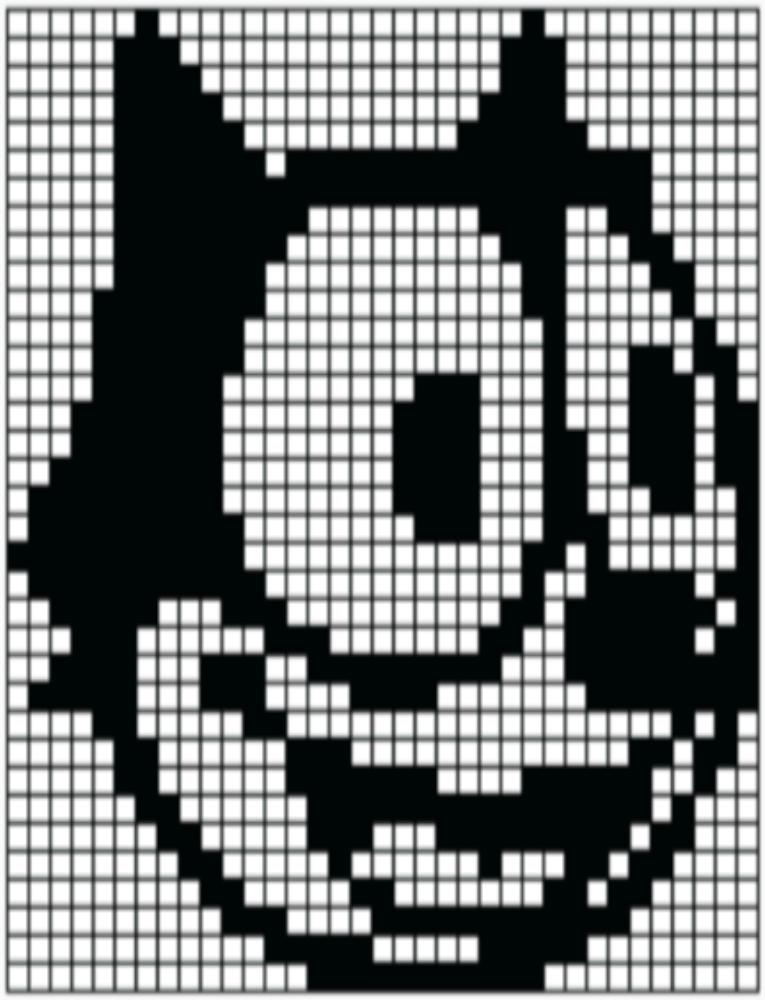
July 11th, 2019

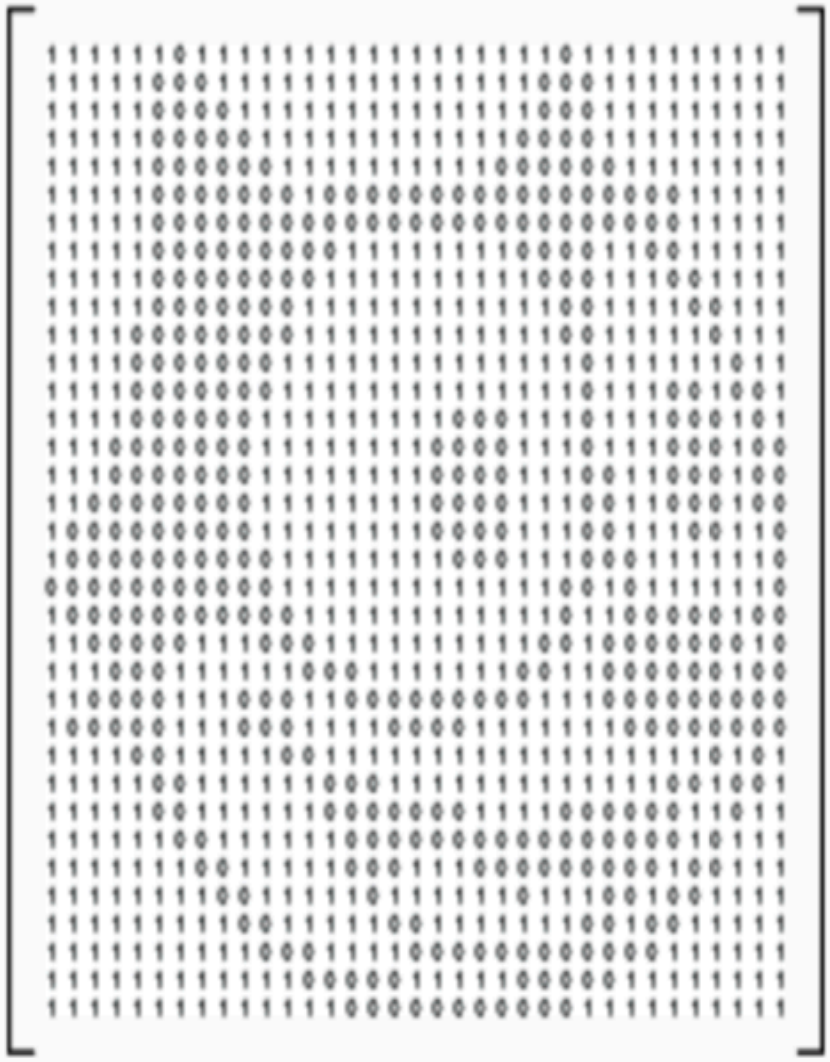
Outline

1. Representing Image Data
2. Convolutional Neural Networks
3. Representing Text Data
4. Recurrent Neural Networks

Representing Image Data

How Do we Represent Gray-scale Images as Numerical Data?





How Do We Represent an Image as a Vector?

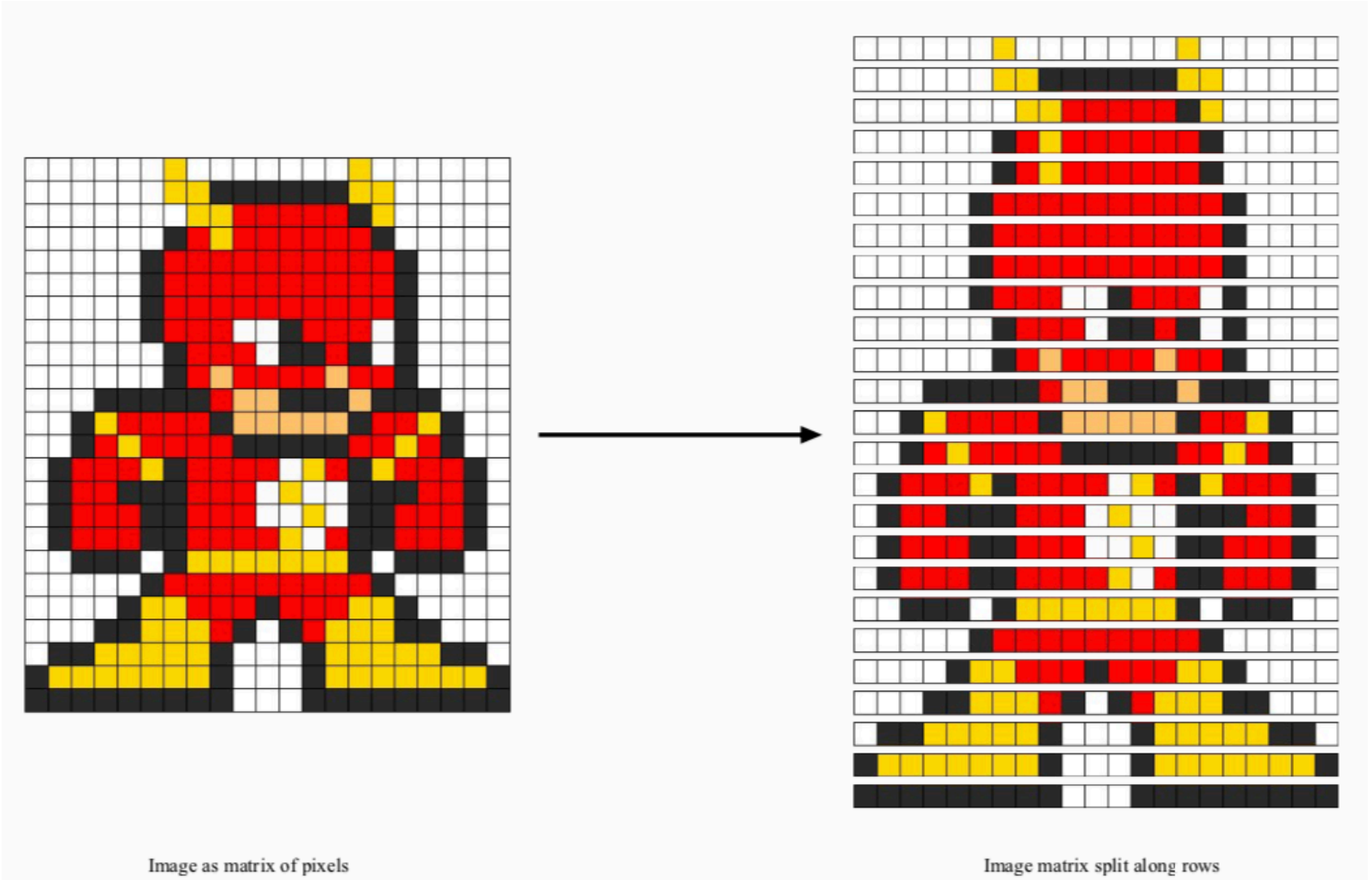
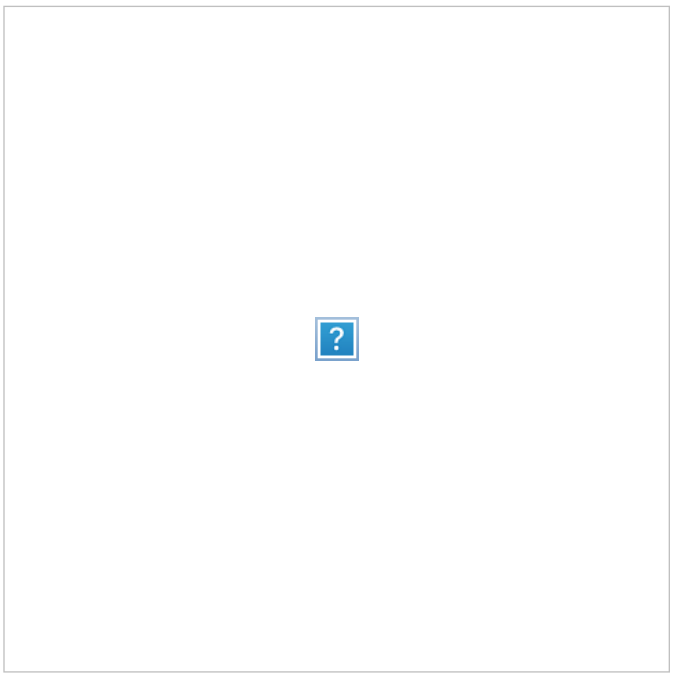


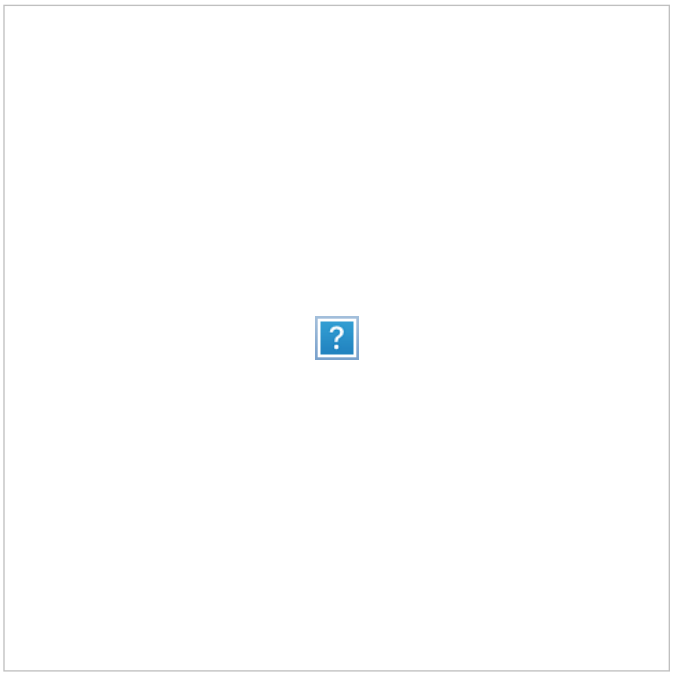
Image as matrix of pixels

Image matrix split along rows

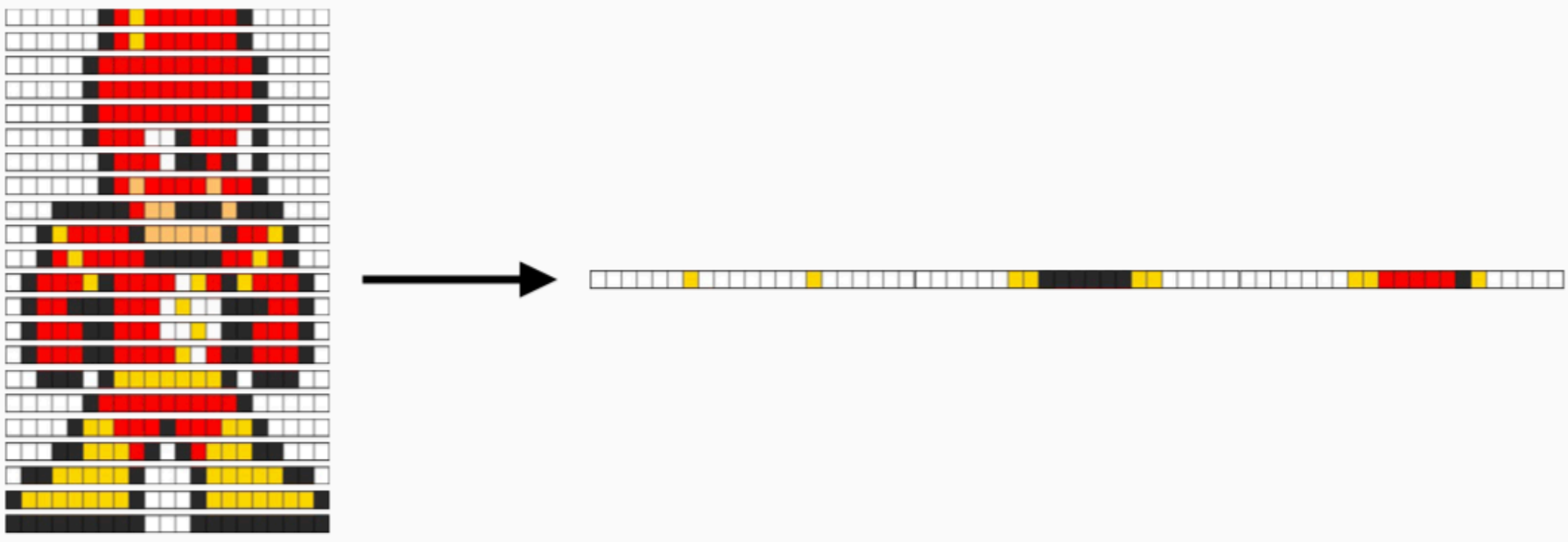
How Do We Represent an Image as a Vector?



How Do We Represent an Image as a Vector?

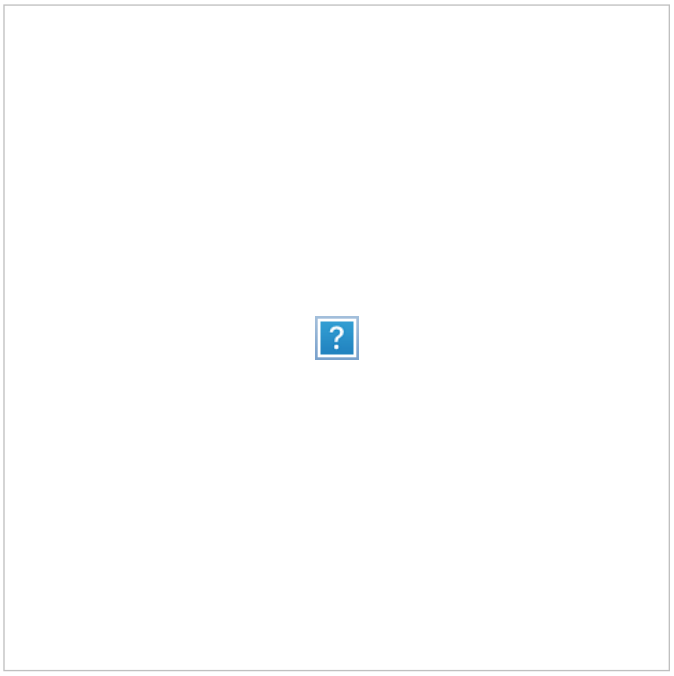


How Do We Represent an Image as a Vector?

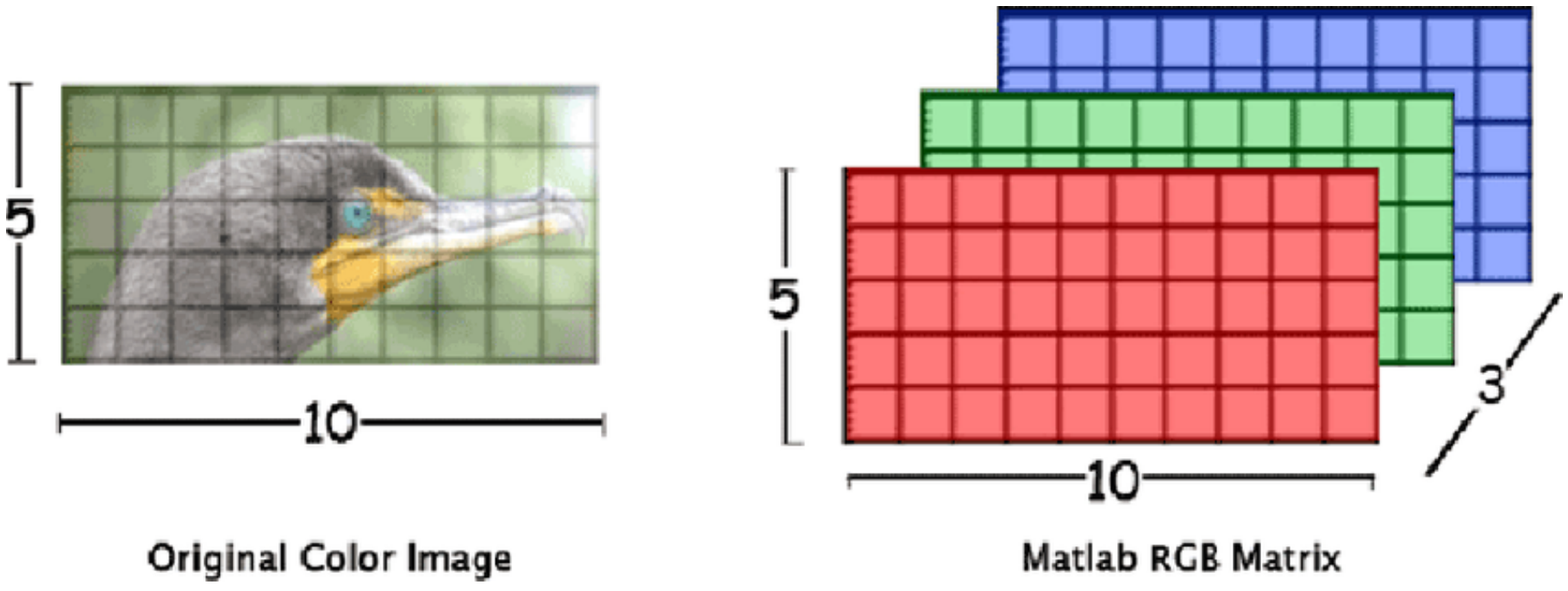


How Do We Represent an Image as a Vector?

This image, when flattened, is represented as a numpy array of shape `(441, )`.



How Do We Represent a Color Image Numerically?



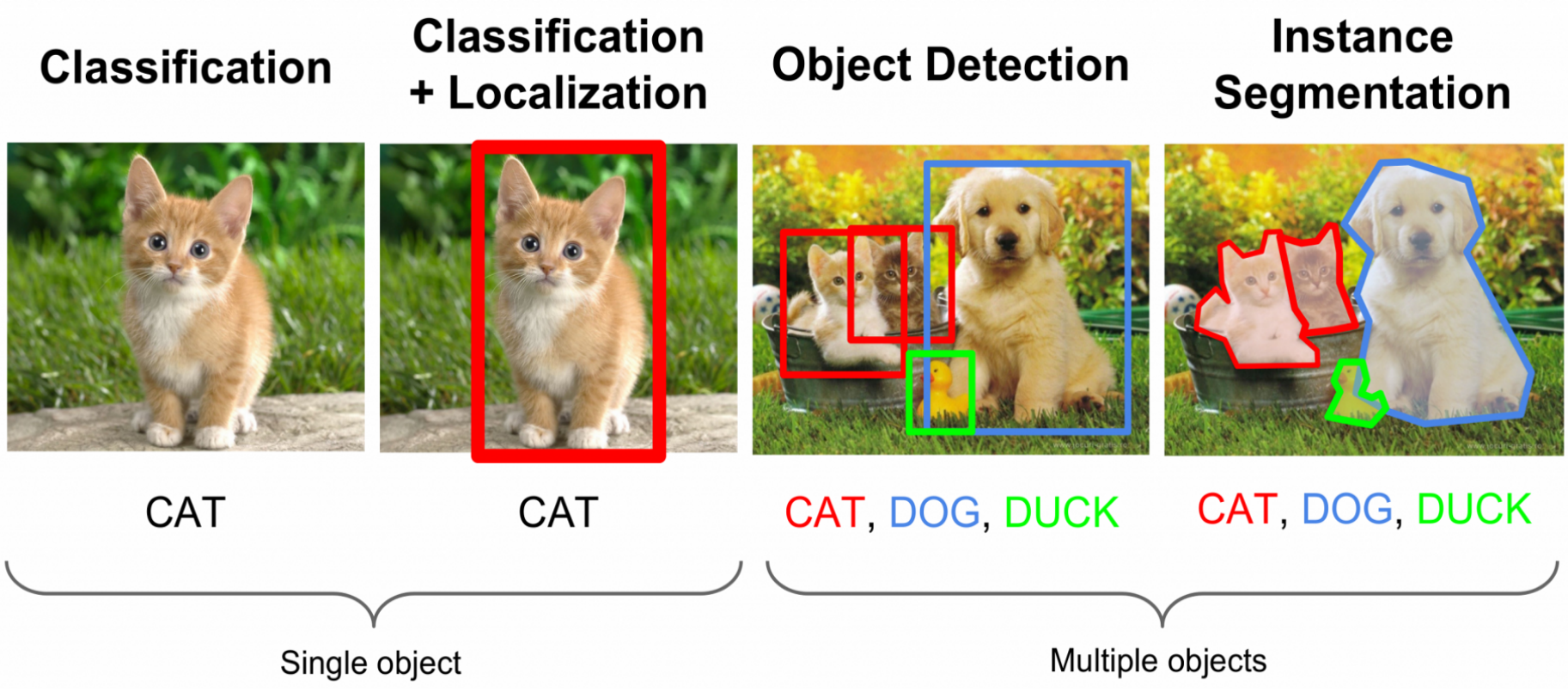
Challenges of Modeling with Images

Working with Image Data is Challenging

In applications involving images, the first task is often to parse an image into a set of 'features' that are relevant for the task at hand. That is, we prefer not to work with images as a set of pixels.

Question: Can you think of why?

Image Data Based Tasks



Feature Extraction with Neural Networks

Goal: find a way to represent images as a set of "features".

Formally, a ***feature***, FF, is an image represented as an array.

We want to learn a function hh mapping an image XX to a set of KK features $[F_1, F_2, \ldots, F_K]$ $[F_1, F_2, \ldots, F_K]$.

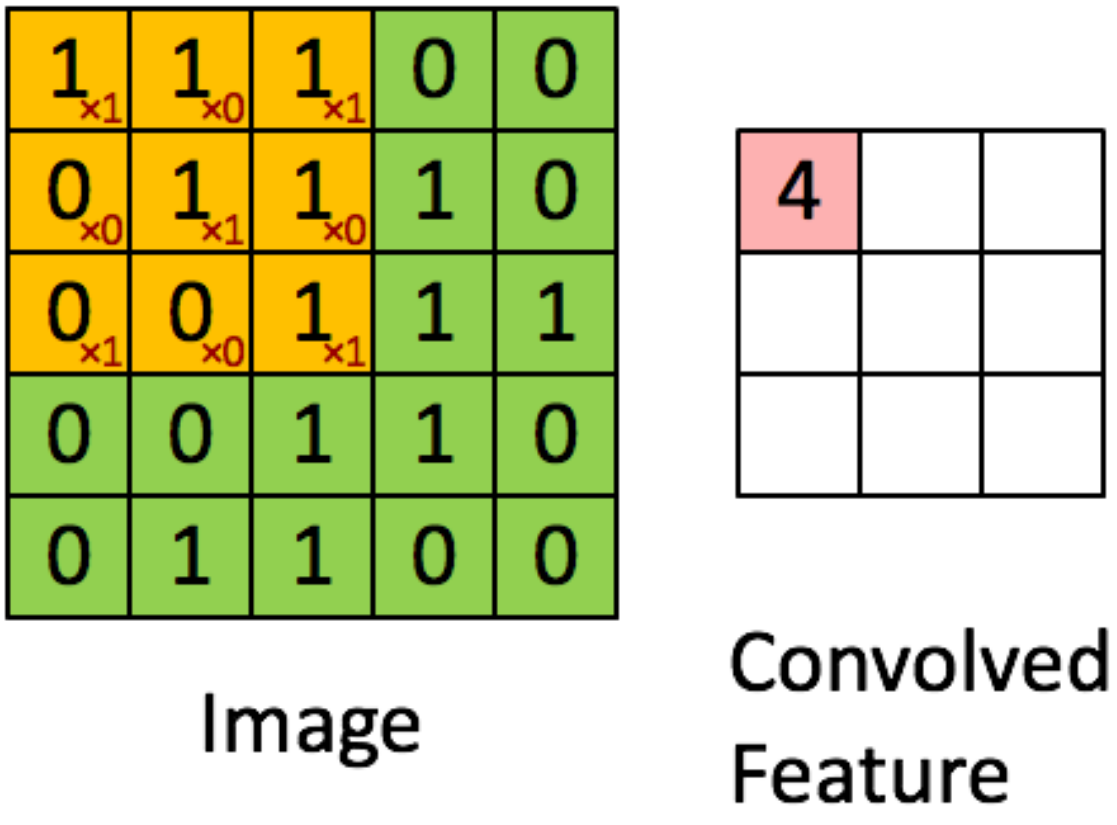
That is, we want to learn a neural network, called a **convolutional neural network**, to represent such a function hh.

Convolutions and Filters

Convolutional Layers

A convolutional neural network typically consists of feature extracting layers and condensing layers.

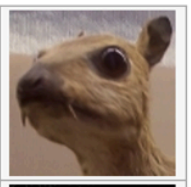
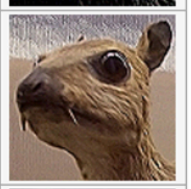
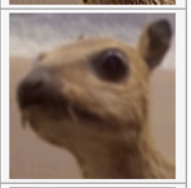
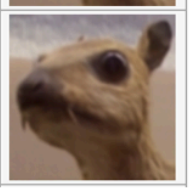
The feature extracting layers are called **convolutional layers**, each node in these layers uses a small fixed set of weights to transform the image in the following way:



This set of fixed weights for each node in the convolutional layer is often called a ***filter*** or a ***kernel***.

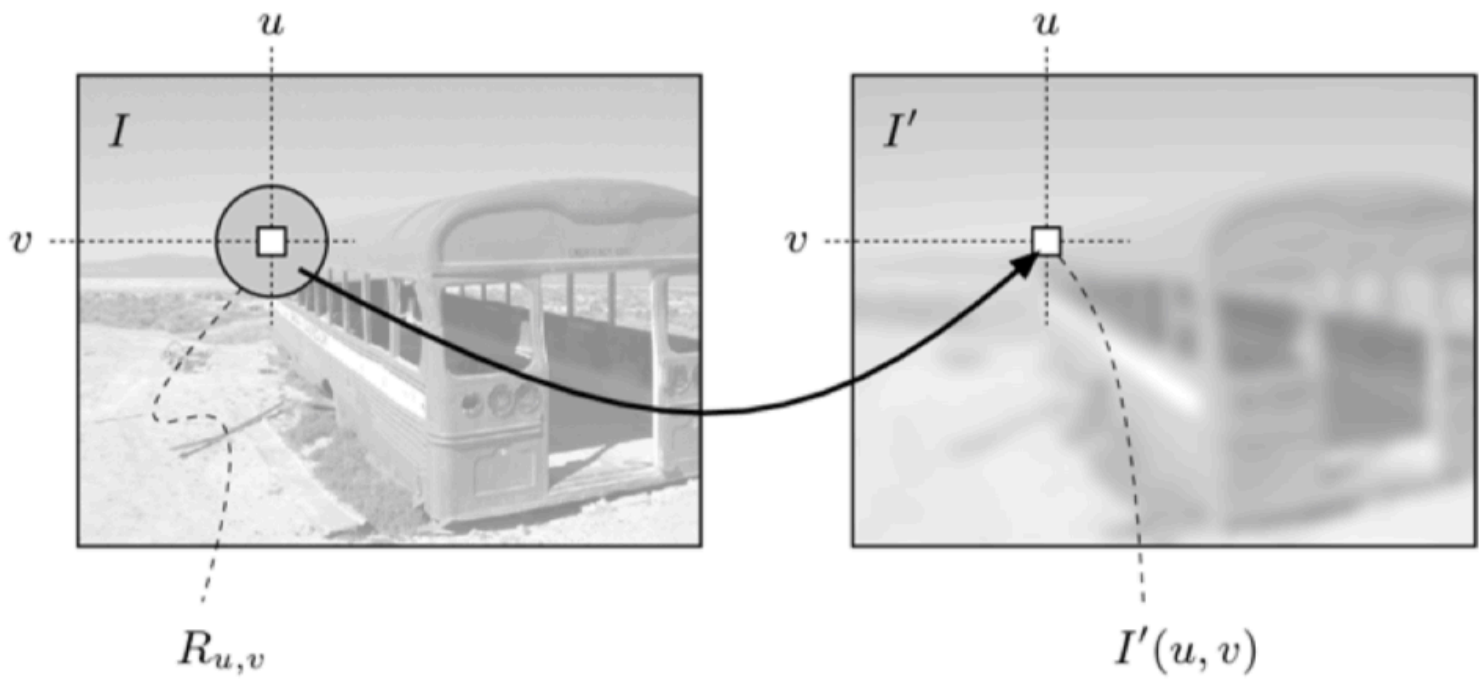
Connections to Classical Image Processing

The term "filter" comes from image processing where one has standard ways to transforms raw images:

Operation	Filter	Convolved Image
Identity	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	
Edge detection	$\begin{bmatrix} 1 & 0 & -1 \\ 0 & 0 & 0 \\ -1 & 0 & 1 \end{bmatrix}$	
	$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$	
	$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$	
Sharpen	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	
Box blur (normalized)	$\frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$	
Gaussian blur (approximation)	$\frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$	

What Do Filters Do?

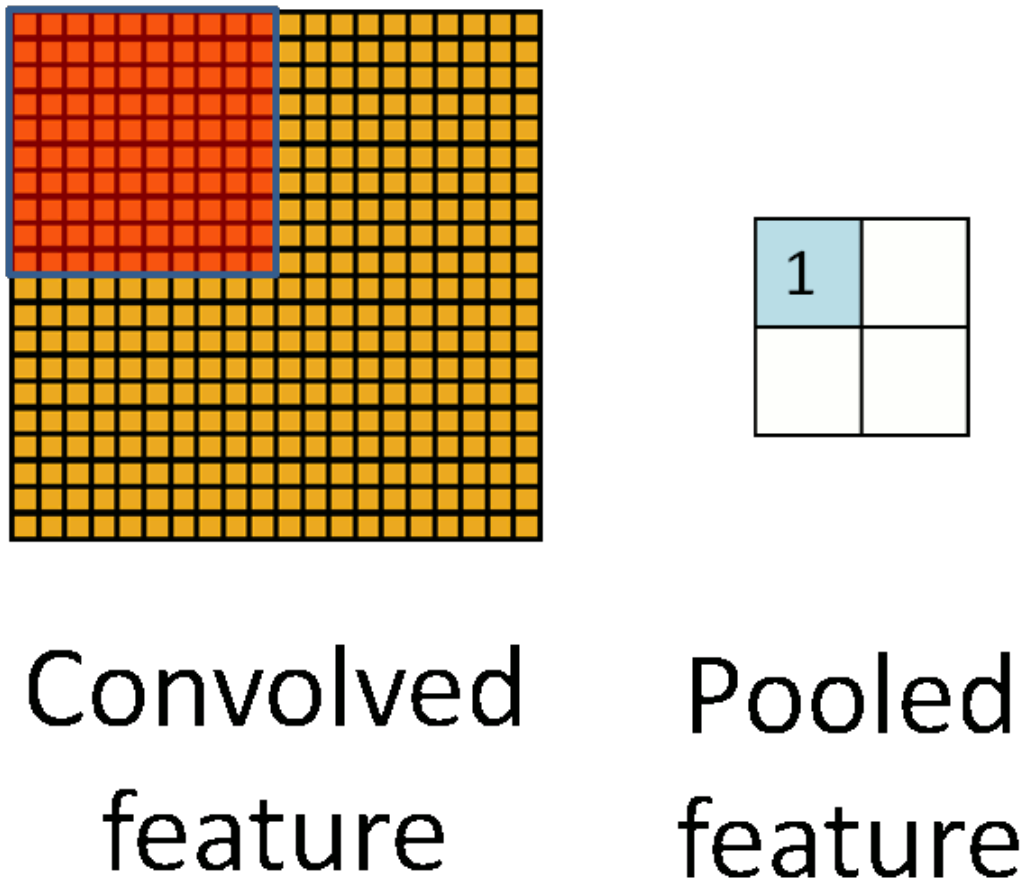
For example, to blur an image, we can pass an $n \times nn \times n$ filter over the image, replacing each pixel with the average value of its neighbours in the $n \times nn \times n$ window. The larger the window, the more intense the blurring effect. This corresponds to the Box Blur filter, e.g. $\frac{1}{9}\begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix}\frac{1}{9}\begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix}$:



In an Gaussian blur, for each pixel, closer neighbors have a stronger effect on the value of the pixel (i.e. we take a weighted average of neighboring pixel values).

Pooling Layers

Often in CNN's we include a **pooling layer** after a convolutional layer. In a pooling layer, we 'condense' small regions in the convolved image:

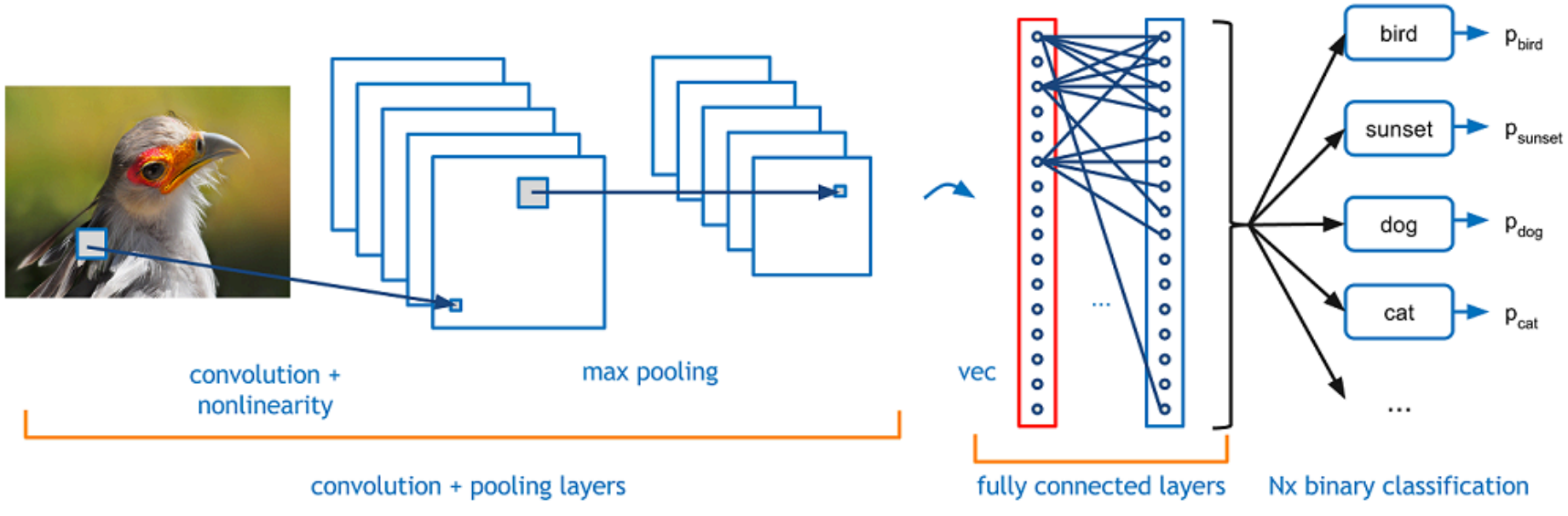


Feature Extraction for Classification

We know that we want to learn the weights of a CNN for feature extraction, but what should our training objective be?

Goal: We should learn to extract features that best helps us to perform our downstream task (classification).

Idea: We train a CNN for feature extraction and a model (e.g. MLP, decision tree, logistic regression) for classification, *simultaneously* and *end-to-end*.



Implementing a Convolutional Neural Network in `keras`

Convolutional Networks for Image Classification

```
# image shape
image_shape = (64, 64)
# Stride size
stride_size = (2, 2)
# Pool size
pool_size = (2, 2)
# Number of filters
filters = 2
# Kernel size
kernel_size = (5, 5)
```

Convolutional Networks for Image Classification

```
cnn_model = Sequential()
# feature extraction layer 0: convolution
cnn_model.add(Conv2D(filters, kernel_size=kernel_size, padding='same',
                     activation='tanh',
                     input_shape=(image_shape[0], image_shape[1], 1)))
# feature extraction layer 1: max pooling
cnn_model.add(MaxPooling2D(pool_size=pool_size, strides=stride_size))

# input to classification layers: flattening
cnn_model.add(Flatten())

# classification layer 0: dense non-linear transformation
cnn_model.add(Dense(10, activation='tanh'))
# classification layer 3: output label probability
cnn_model.add(Dense(1, activation='sigmoid'))

# Compile model
cnn_model.compile(optimizer='Adam',
                  loss='binary_crossentropy',
                  metrics=['accuracy'])
```

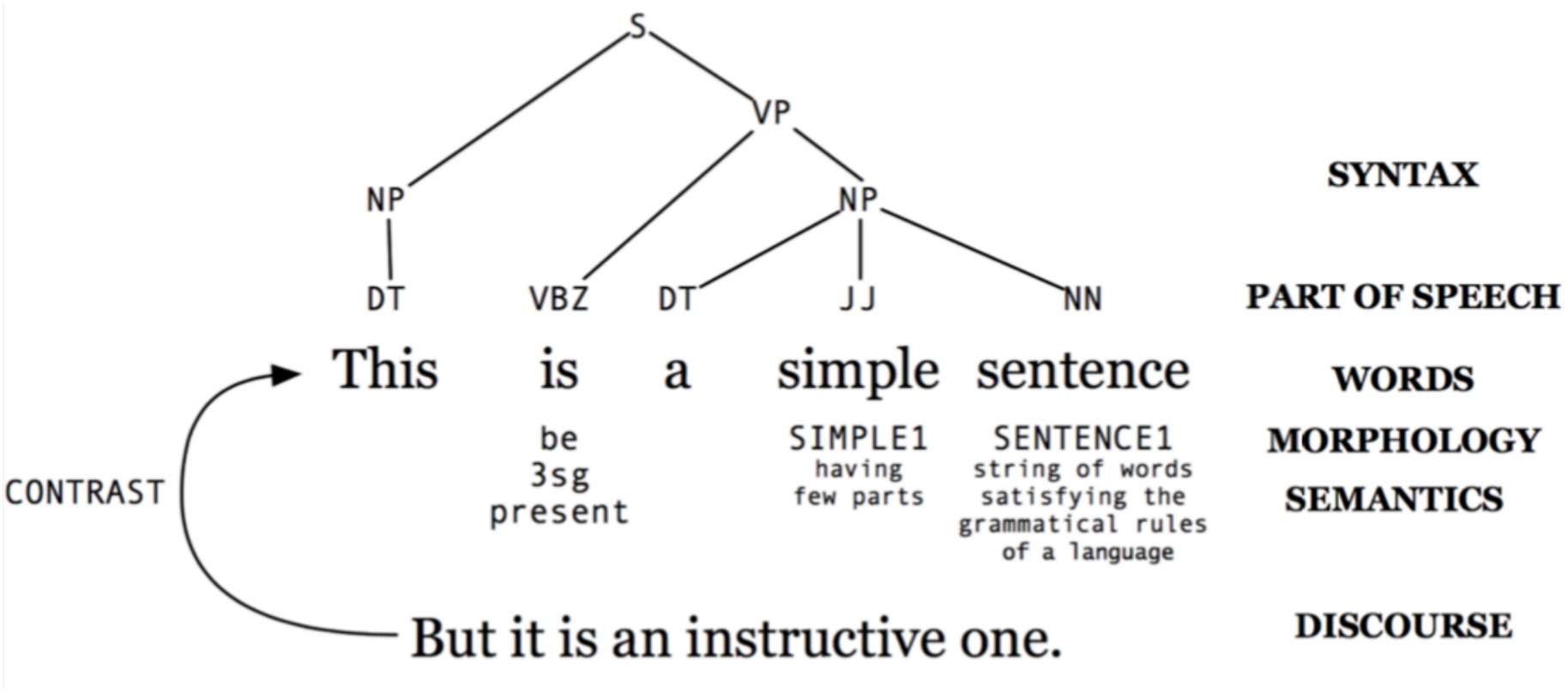
Using Pre-Trained CNNs with Any Classifier

You can use a number of pretrained CNNs for feature extraction (https://keras.io/applications/) (https://keras.io/applications/)), and then use these features as input for any classifier (e.g. random forest, decision tree, MLP).

What is Natural Language Processing?

Levels of Linguistic Knowledge in NLP

Natural language processing deals with building models/algorithms to automatically analyze and represent human language.



NLP: Tasks and Applications

Tasks: do this...

1. Classify entire texts
2. Classify individual words
    - Parts of speech tagging
    - Chunking
    - Parsing/stemming
    - Semantic rolel abeling
3. Generating text
    - Speech recognition
    - Machine translation
    - Summarization

Applications: in order to...

1. Classify document: spam, sentiment, etc
2. Auto-complete and auto-correct
3. Build conversational agents/dialogue systems

Main Challenges

1. Ambiguity all all levels: ‘I made her duck’, ‘I went to the bank...’
2. Language changes through time, across domains
3. Information retrieval
4. Many rare words

Typical Approach

All NLP solutions involve three phases:

1. Create a representation of the text
2. Extract ‘important features’
3. Build a (statistical) machine learning model to accomplish task using these features

Traditionally, the features are manually and task-specifically engineered. More recently, task-agnostic ways of learning ‘important features’ have become possible with highly flexible models like neural networks.

How Do We Represent Text

Representing Textual Data

Comparing the content of the following two sentences is easy for an English speaking human (clearly both are discussing the same topic, but with different emotional undertone):

1. Linear R3gr3ssion is very very cool!
2. What don’t I like it a single bit? Linear regressing!

But a computer doesn’t understand

- which words are nouns, verbs etc (grammar)
- how to find the topic (word ordering)
- feeling expressed in each sentence (sentiment) We need to represent the sentences in formats that a computer can easily process and manipulate.

Preprocessing

If we're interested in the topics/content of text, we may find many components of English sentences to be uninformative.

1. Word ordering
2. Punctuation
3. Conjugation of verbs (go vs going), declension of nouns (chair vs chairs)
4. Capitalization
5. Words with mostly grammatical functions: prepositions (before, under), articles (the, a, an) etc
6. Pronouns?

These uninformative features of text will only confuse and distract a machine and should be removed.

Representing Documents: Bag Of Words

After preprocessing our sentences:

1. (**S1**) linear regression is very very cool
2. (**S2**) what don't like single bit linear regression

We represent text in the format that is most accessible to a computer: numeric.

We simply make a vector of the counts of the words in each sentence.

	linear	regression	is	very	cool	what	don't	like	single	bit
S_1	1	1	1	2	1	0	0	0	0	0
S_2	1	1	0	0	0	1	1	1	1	1

Turning a piece of text into a vector of word counts is called ***Bag of Words***.

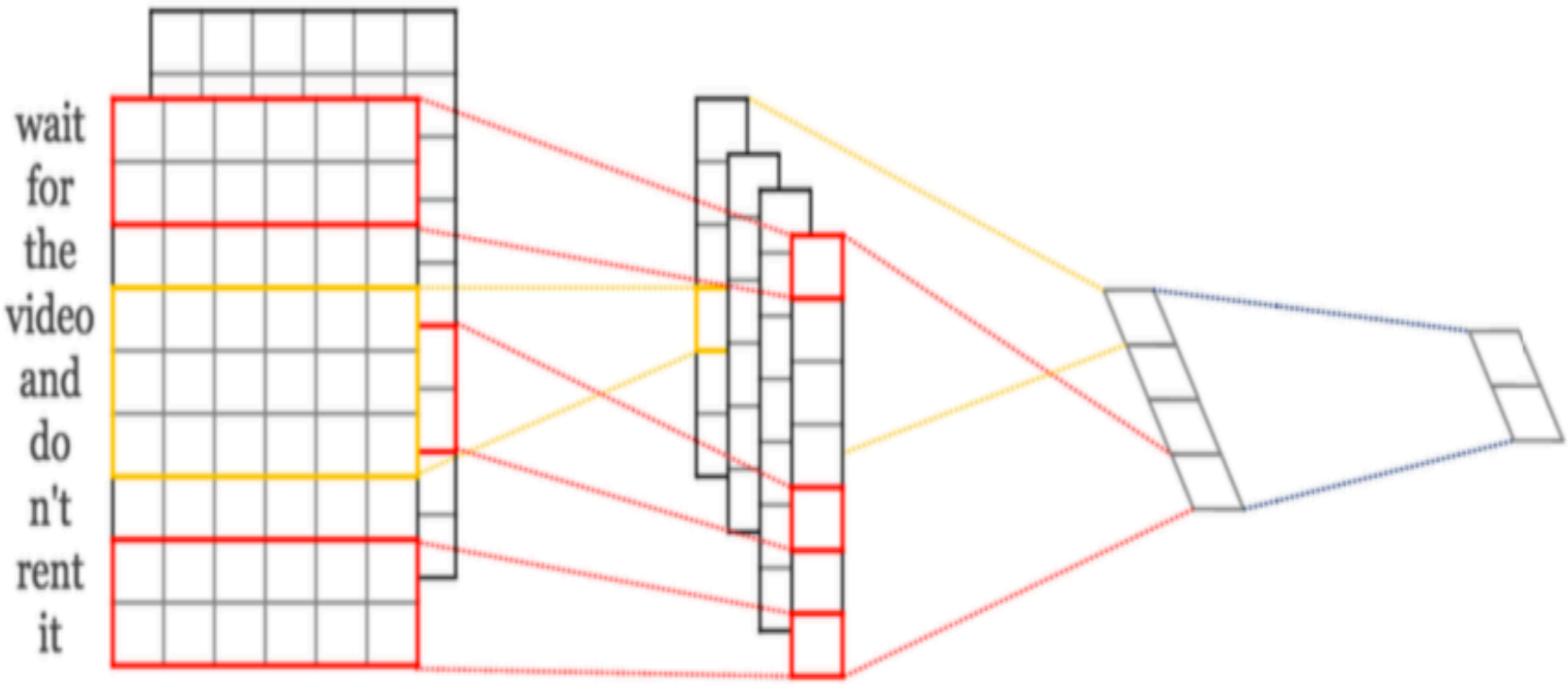
Bag of Words Representation in `python`

```
from sklearn.feature_extraction.text import CountVectorizer

# define documents
corpus = ['Well done!', 'Good work, good!',  'Excellent!', 'Poor effort!',
'not good', 'poor work', 'Could have done better on this.']

# vectorize text
vectorizer = CountVectorizer(stop_words=['on', 'this'], min_df=0., max_df=1
.)
x = vectorizer.fit_transform(corpus).toarray()
```

Document Classification Using Bag of Words



What Do the Hidden Layers in the Network Mean?

The hidden layers of the document classifying are representations of words are **low-dimensional** real-valued vectors. These vectors are called ***word embeddings***. Sometimes, these representation captures semantic information!