Applied Machine Learning for Business Analytics

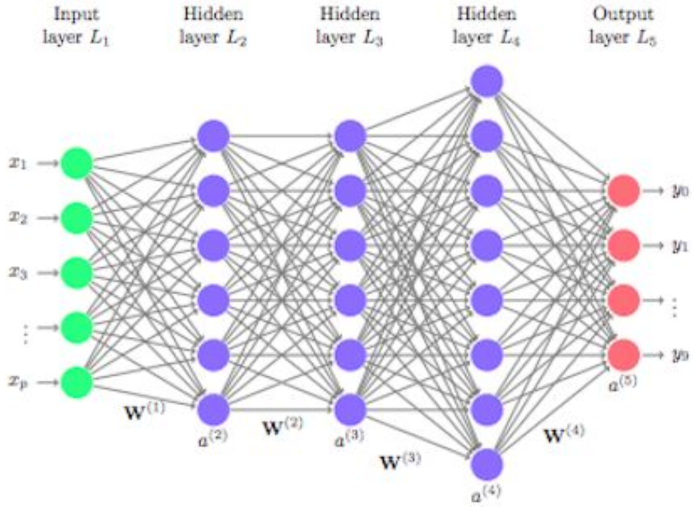
Lecture 6: Convolutional Neural Network

Lecturer: Zhao Rui

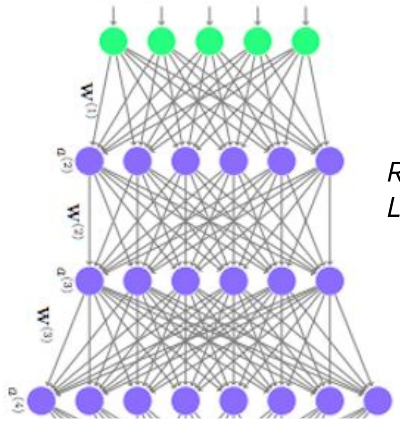
Agenda

1. Introduction to CNN
2. Why CNN for images?
3. Limitations of CNN

Hidden representation in deep learning



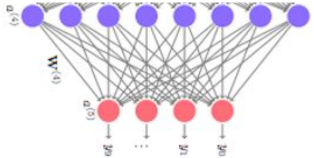
Low-dim, Original Space



Representation Learning

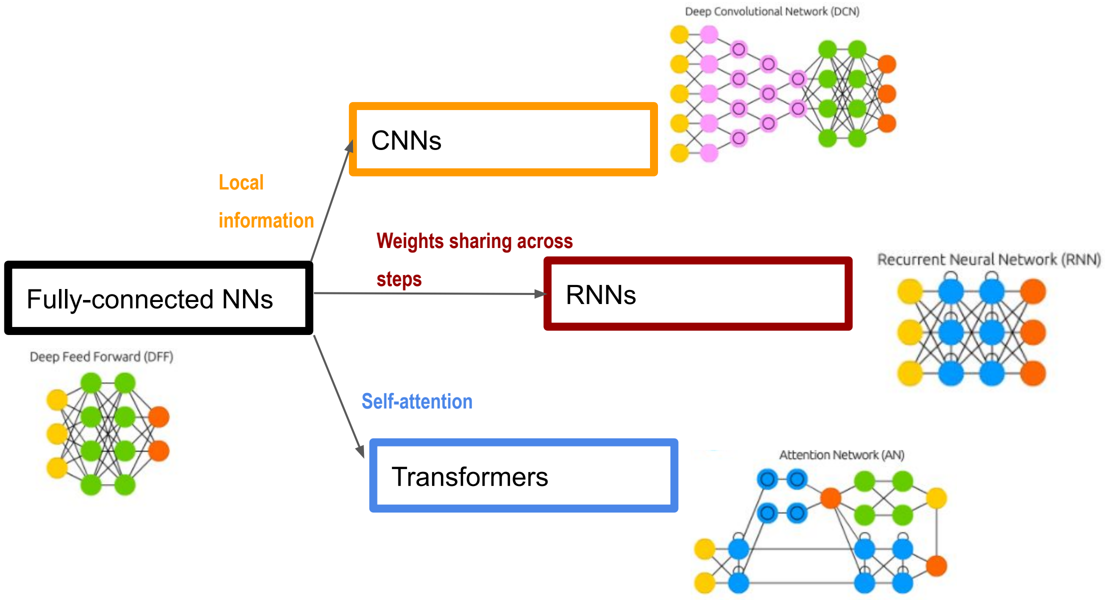
High-dim, **Linearly Separated** Space

Classification



Softmax Classifier (Linear Model)

Deep learning structures



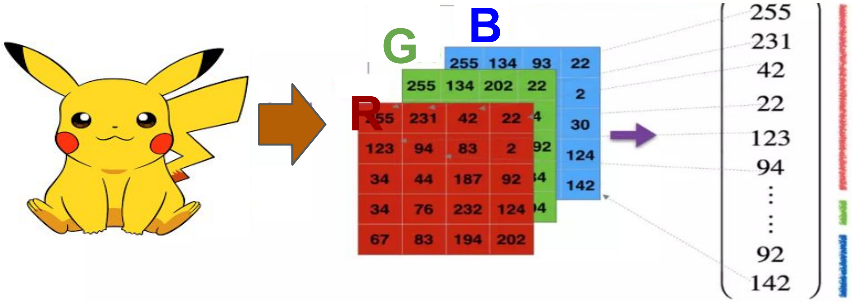
https://www.asimovinstitute.org/author/fjodorvanveen/

1. Introduction to CNN

Image: a matrix of pixel values

- Every image can be represented as a matrix of pixel values
- The pixel value ranges from 0 to 255.
- Channel is referred to a certain component of an image
  - An image from your iphone will have three channels
  - A grayscale image has just one channel

Computers see image



Softmax Classifier
SVM
….

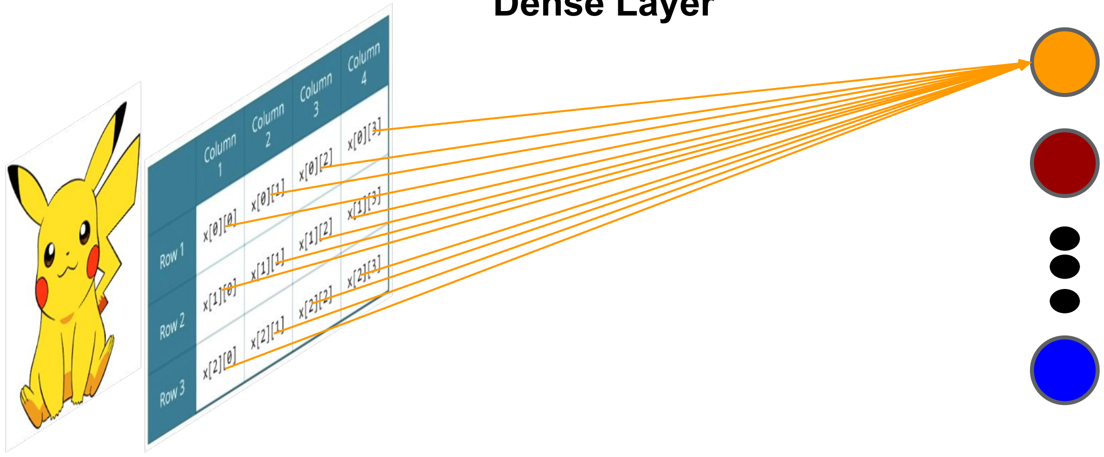
Fully-Connected
Neural Network

Labels

Is Dense layer a good feature extractor for Images?

Hidden Layer

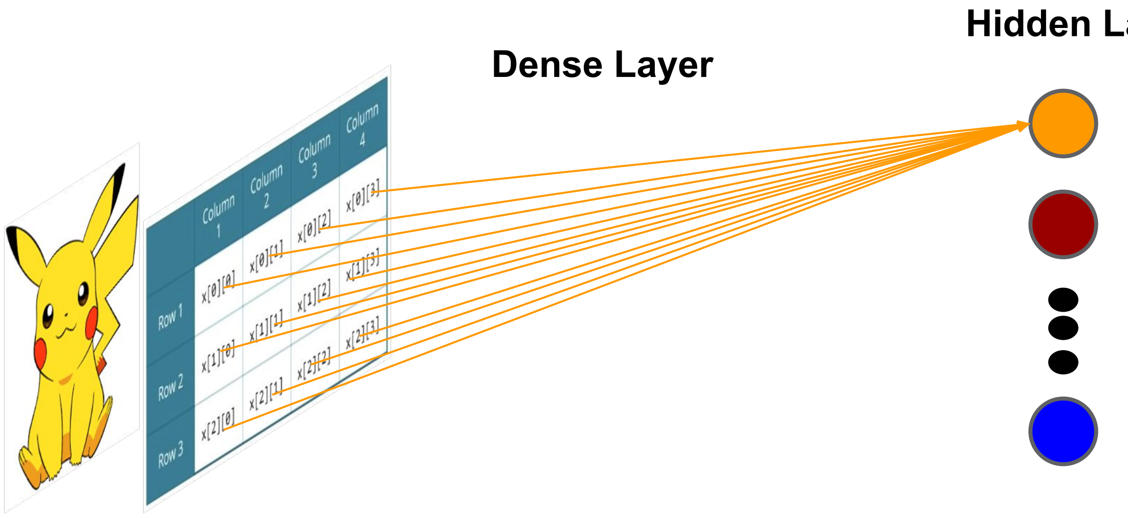
Dense Layer



What will each neuron learn from the image?

Input Image as a matrix
(assume it is a grayscale one)

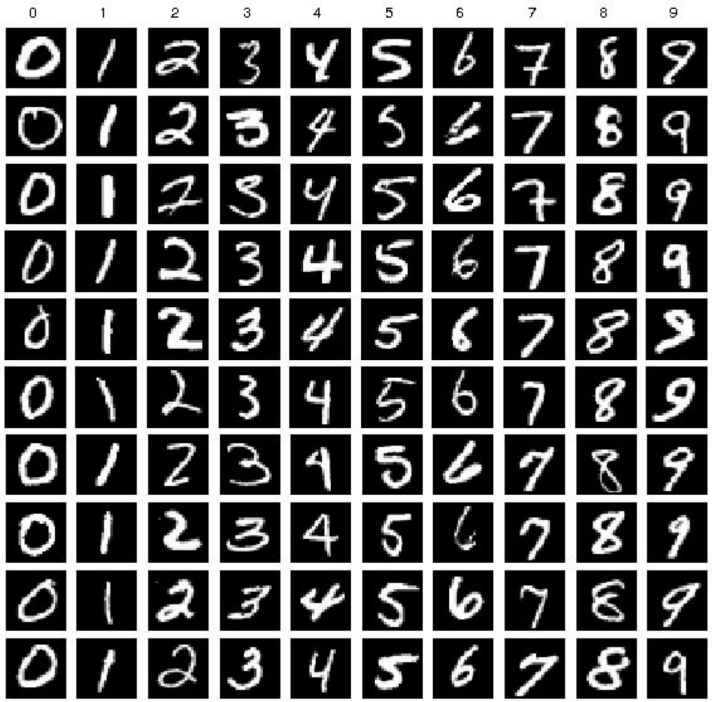
Is Dense layer a good feature extractor for Images?



Input Image as a matrix
(assume it is a grayscale one)

Each Neuron is the weighted sum of all pixel values in that image plus non-linear transformation

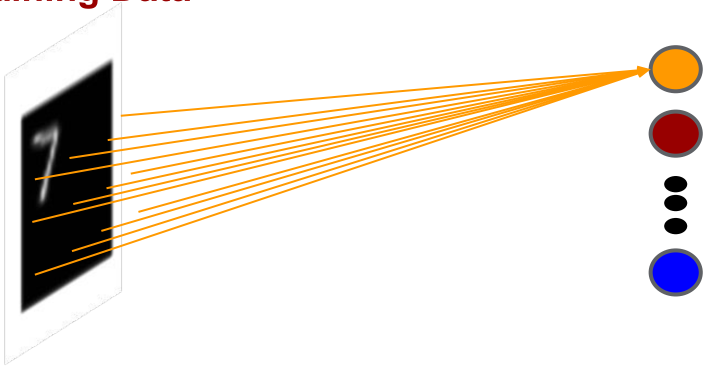
Think about MNIST dataset



The above model requires the digit should be in the center of the image and it had to be the only thing in the image.

What if the digits in top-left corner

Training Data



Testing Data

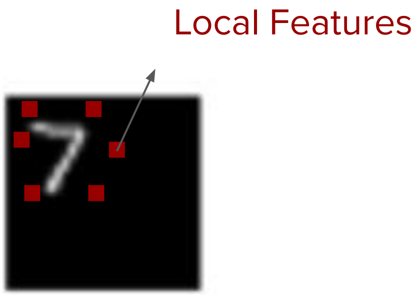
The model will generate two different vectors even two data samples belong to the same class

Limitations of fully-connected neural networks

- For the grayscale image is 64 pixel by 64 pixel
- Image is represented by 64 * 64 * 1 = 4096 values
- FCNN’s input size is 4096
- If the first hidden layer size is 500,
  - Number of weights in the first hidden layer is 4096*500 = 2,048,000
- The model size will explode further
  - Deep structures (many layers)
  - Color images (the input size will be 3 times)
- The concern for a huge model size:
  - Risk of Overfitting
  - Make training/deployment more time/resource consuming
  - Make learning more untraceable as dimension of search space is increased.

Limitations of fully-connected neural networks

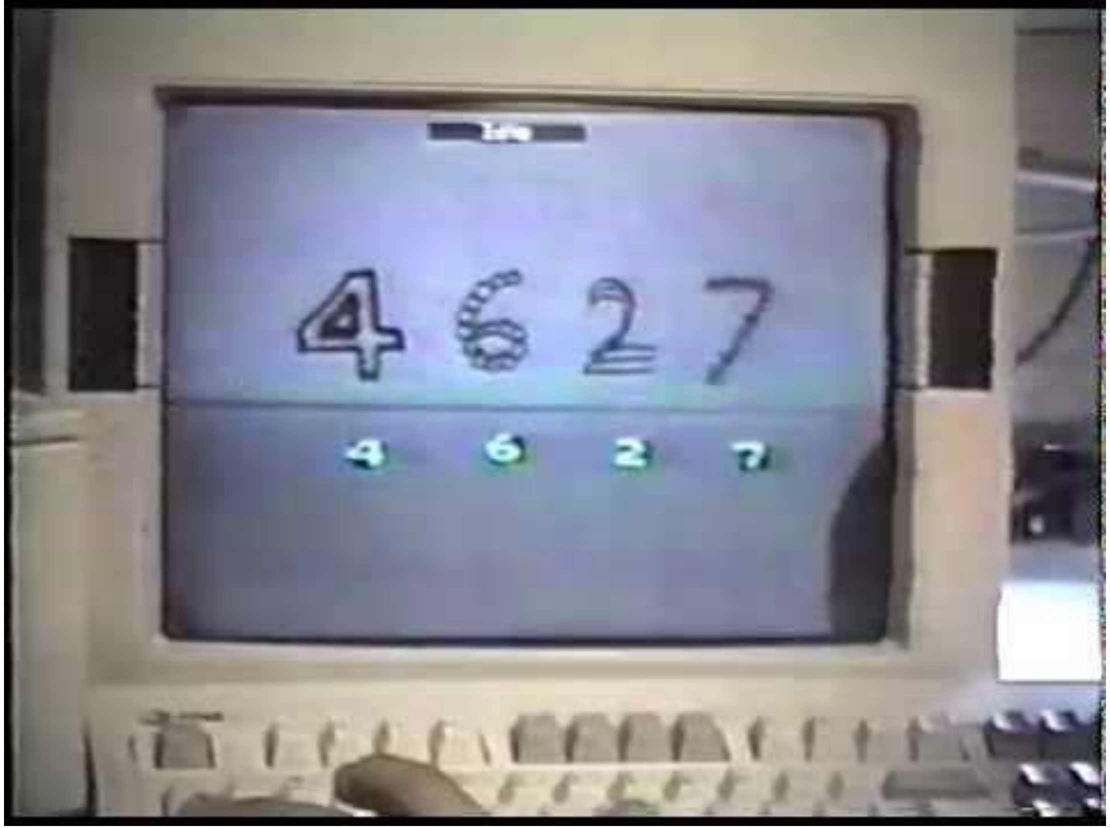
- FNN can not scale easily to computer vision (Input Size is so big-> too many weights)
- Any spatial relationship is not captured
  - 2D image is flattened to be a 1D vector.
- Global Pattern vs Local Pattern
  - In FNN, each pixel in the image is connected to the hidden neuron
  - The hidden neuron tries to learn the “global feature”



Cat vs dog

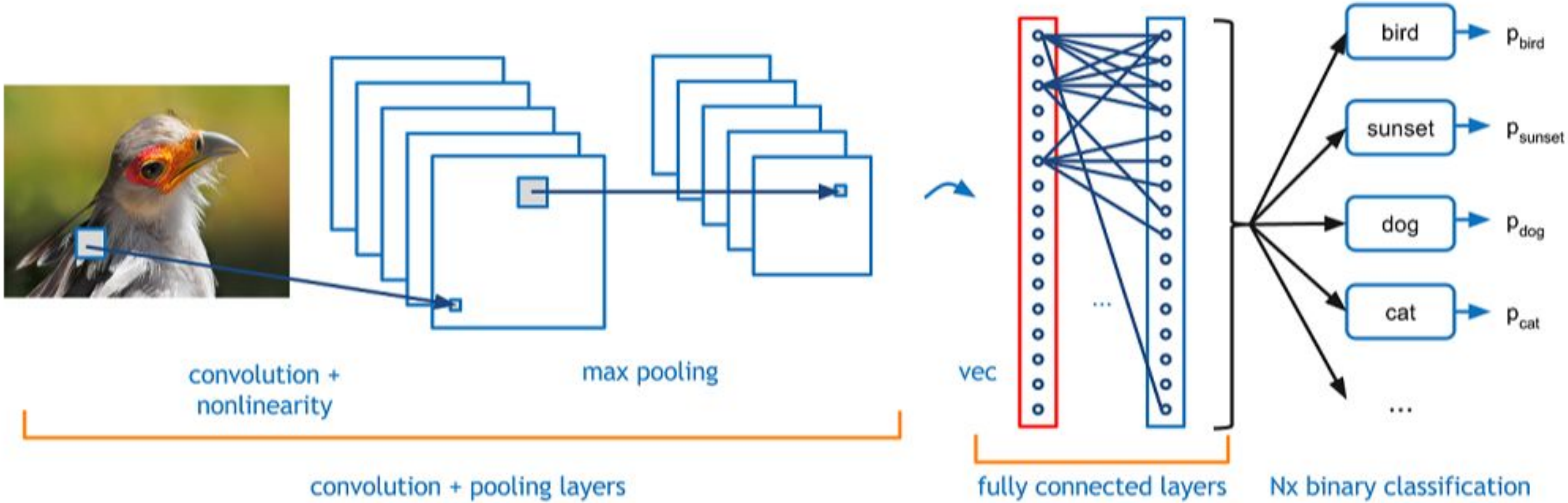
- To recognize those images, we captures the patterns
- For Cat vs Dog problems, patterns can be
  - Shapes of ears, eyes
  - Colors
  - Hairs
- Machine learning model should be trained to capture those **patterns**

In the context of CNN, patterns are also called as kernel, filters and feature extractor.



https://www.youtube.com/watch?v=FwFduRA_L6Q

Convolutional neural network

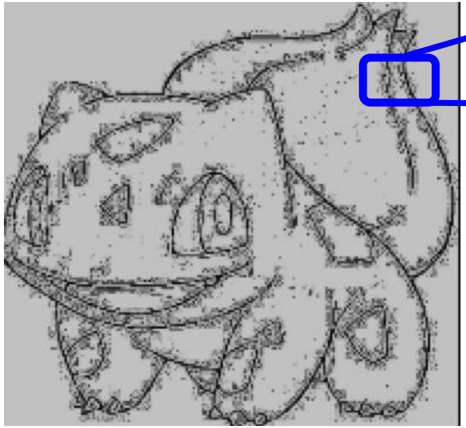


Extracting useful features of data

Perform a ML task (like classification based on the vectorized data)

Conv-operation

Consider neighbor values



1	0	1
0	$\boxed{1}$	0
1	0	1

Current Pixel Value is 1

Dot Product

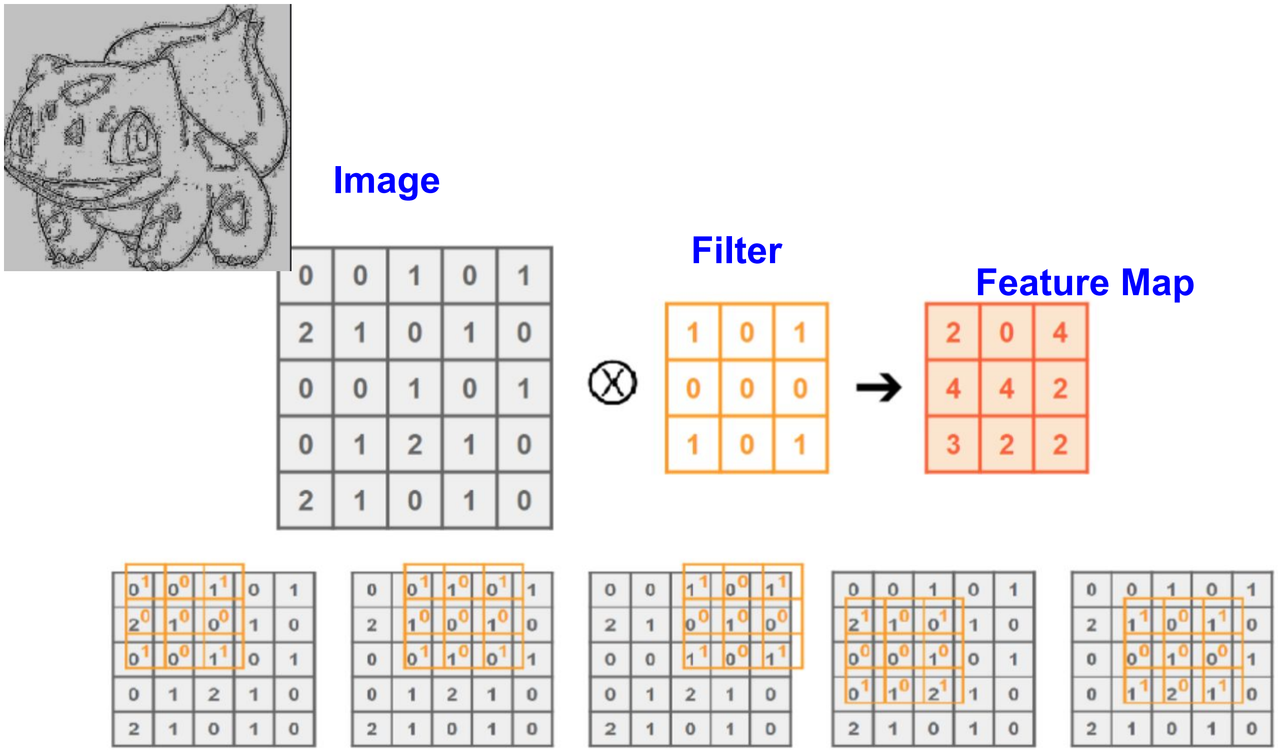
1	**0**	**1**
0	**0**	**0**
1	**0**	**1**

Filter (3 by 3)

New Pixel Value
4

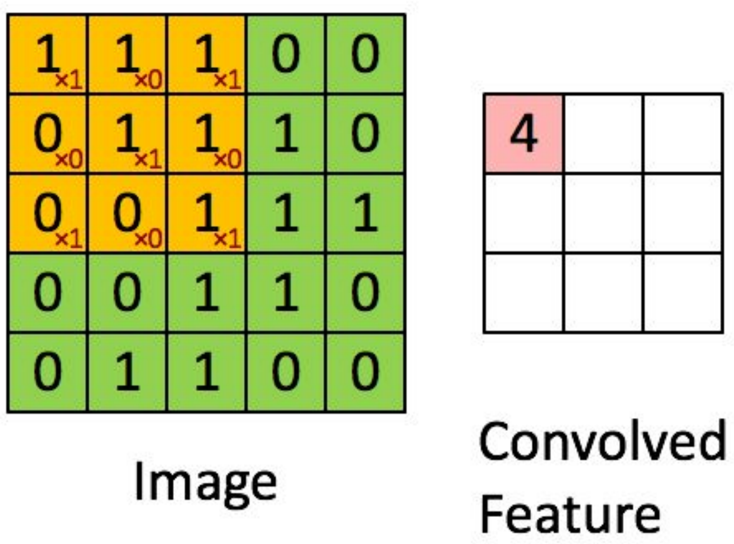
Local Patterns

Conv-operation



Conv-operation

- Apply the same filter for every pixel in the original image
- Filter size is the shape of the filter matrix (yellow one)



Feature Map

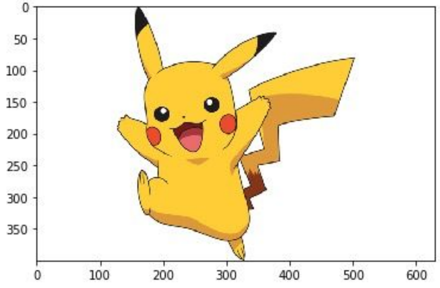
Check gif version here:
https://docs.google.com/presentation/d/1V_7IqLDsKXyaEwR9ZgxmlQ9ixmcT41ZGOLmJtbpgGPM/edit?usp=sharing

Stanford UFLDL

Conv-operation

- Convolution is a mathematical operation on two objects to product an outcome that expresses how the shape of one is modified by the other
- In the CNN, the feature map has the information about the particular pattern corresponding to the filter

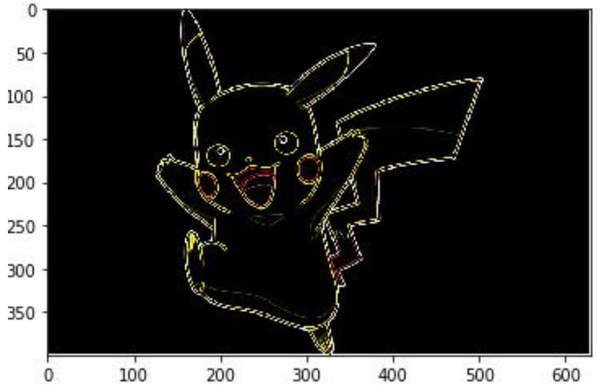
Feature map



Image

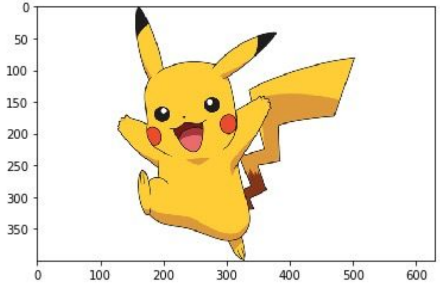
```
print(kernel)

[[-1 -1 -1]
 [-1  8 -1]
 [-1 -1 -1]]
```

Edge Detection



Feature Map

Feature map

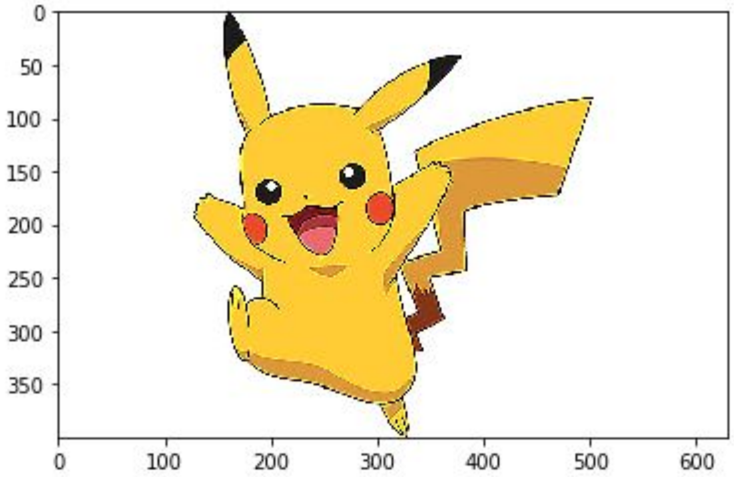


Image

```
print(kernel)
```

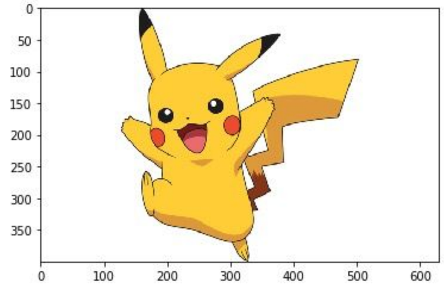
```
[[ 0 -1  0]
 [-1  5 -1]
 [ 0 -1  0]]
```

Sharpen



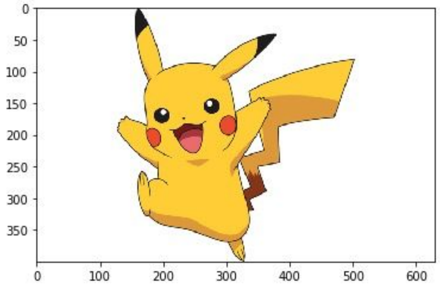
Feature Map

Feature map



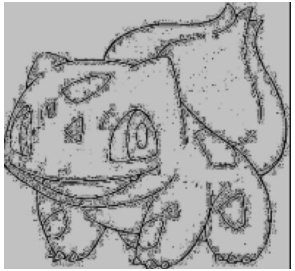
Image

Identity



Feature Map

Conv-operation



Image

0	0	1	0	1
2	1	0	1	0
0	0	1	0	1
0	1	2	1	0
2	1	0	1	0

$\otimes$

Filter

1	0	1
0	0	0
1	0	1

$\rightarrow$

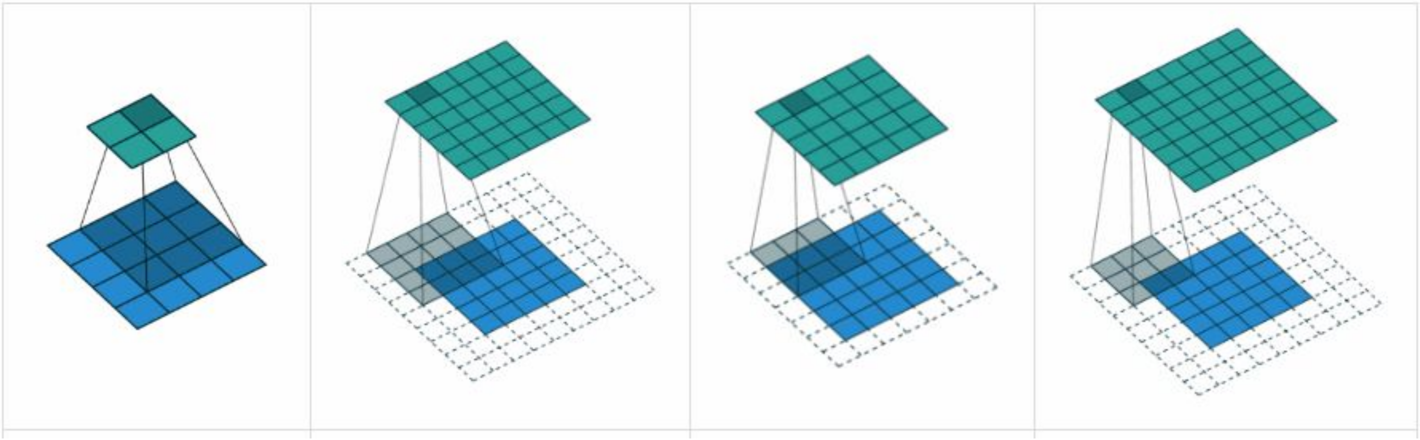
Feature Map

2	0	4
4	4	2
3	2	2

Those edge pixels are not captured

Padding

- Padding: give additional pixels around the boundary of the image
- Padding size: the number of additional pixels



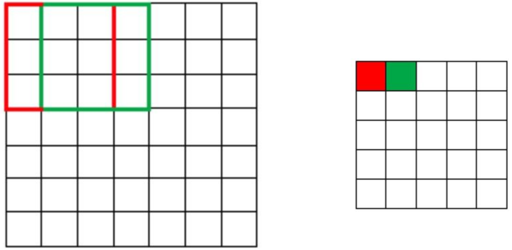
Padding Size: 0
Valid

Padding Size: 1
Same

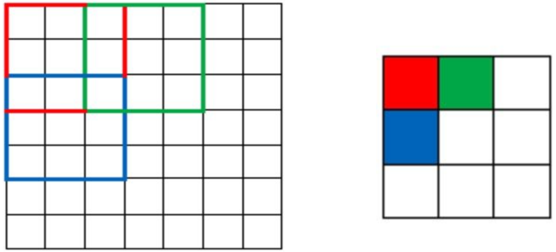
Stride size

- Does a filter always have to move one pixel at a time?
- Stride size is the amount by which the filter shifts

Stride is 1



Stride is 2



Convolutional operation

- Three conv. Layer basic hyper-parameters:
  - Filter size: K
  - Stride size: S
  - Padding size: P
- Output Size can be decided by

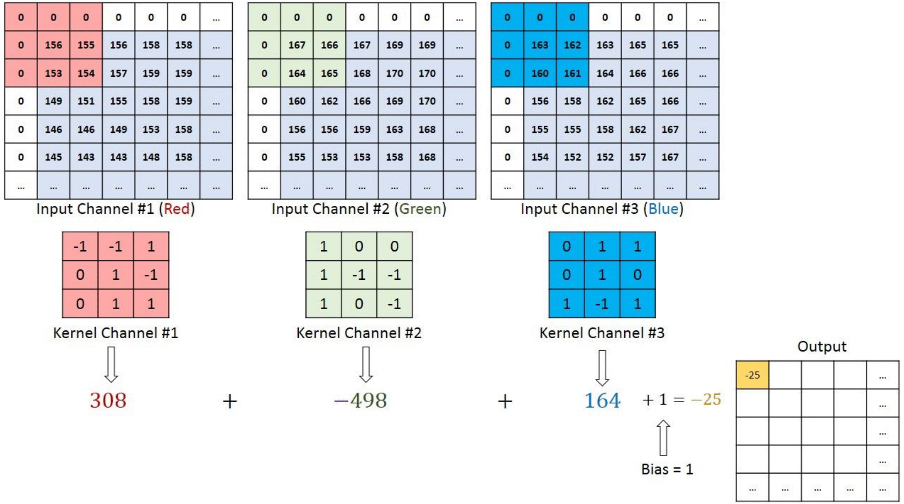
Input size

$$o = \frac{W - K + 2P}{S} + 1$$

Output size

Multi-Channel CNN

- A color image is a 3-D tensor
- 400 (height) 630 (width) 3 (R,G,B channels)



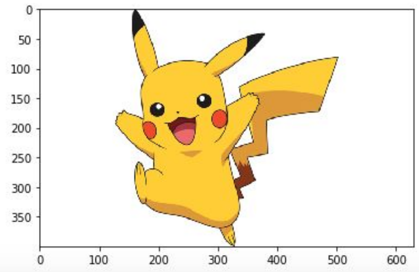

```python
from matplotlib.image import imread
import numpy as np
img = imread('pikka_3.jpg')
```

```python
print(img.shape)
```

```
(400, 630, 3)
```

```python
plt.imshow(img, interpolation='nearest')
```

```
<matplotlib.image.AxesImage at 0x11b404278>
```



From Keras Layers Conv2D

Input shape

4D tensor with shape: `(batch, channels, rows, cols)` if `data_format` is `"channels_first"` or 4D tensor with shape: `(batch, rows, cols, channels)` if `data_format` is `"channels_last"`.

Output shape

4D tensor with shape: `(batch, filters, new_rows, new_cols)` if `data_format` is `"channels_first"` or 4D tensor with shape: `(batch, new_rows, new_cols, filters)` if `data_format` is `"channels_last"`. `rows` and `cols` values might have changed due to padding.

https://www.researchgate.net/post/How_will_channels_RGB_effect_convolutional_neural_network

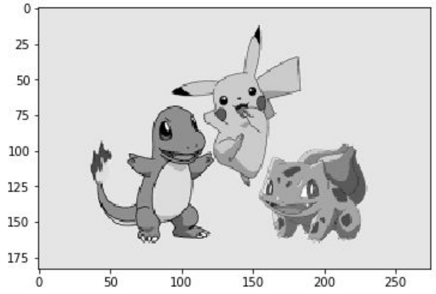
Where are these filters from?

- Filters, in nature, are model parameters, which can be **learned** by Gradient Descent Algorithms .
- These filters weights are firstly randomly initialized, and then updated during training process.
- End-to-End optimization: Gradients computed by backpropagation.
- More details:

https://towardsdatascience.com/training-a-convolutional-neural-network-from-scratch-2235c2a25754

Non–linear activation

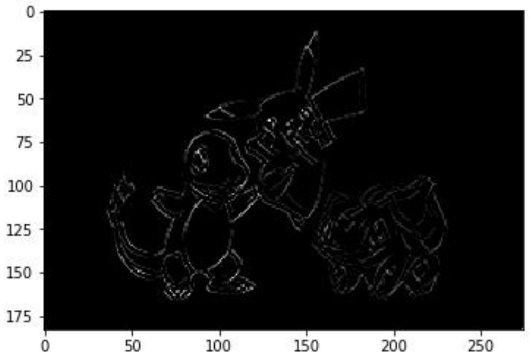
- Filter operation is dot product (linear computation)
- In deep learning, we need to have non-linear transformations
- Add non-linear activation



Image

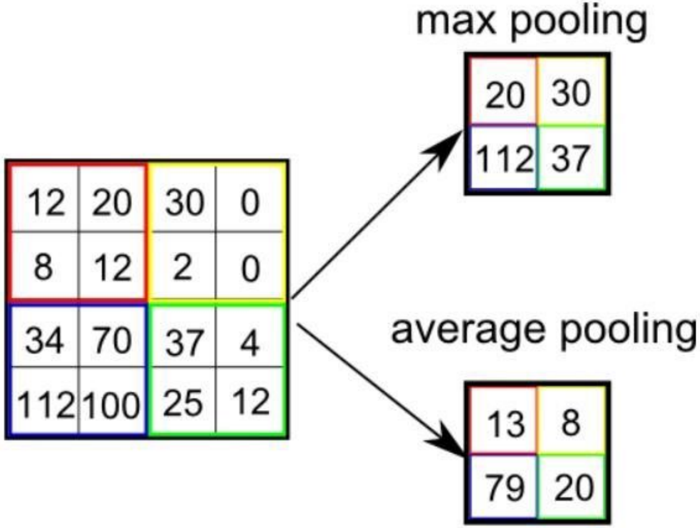
```
print(kernel)
[[ 1  0 -1]
 [ 0  0  0]
 [-1  0  1]]
```

non-linear



Pooling operation

- Pooling Size: the box size. Here is 2 by 2
- Reduce the dimensionality
- Remove some noise
- Extract significant values

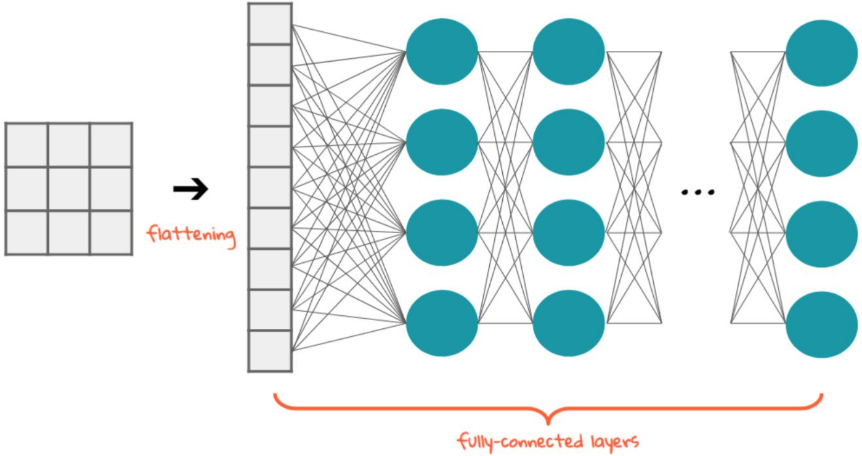


Filter then pool

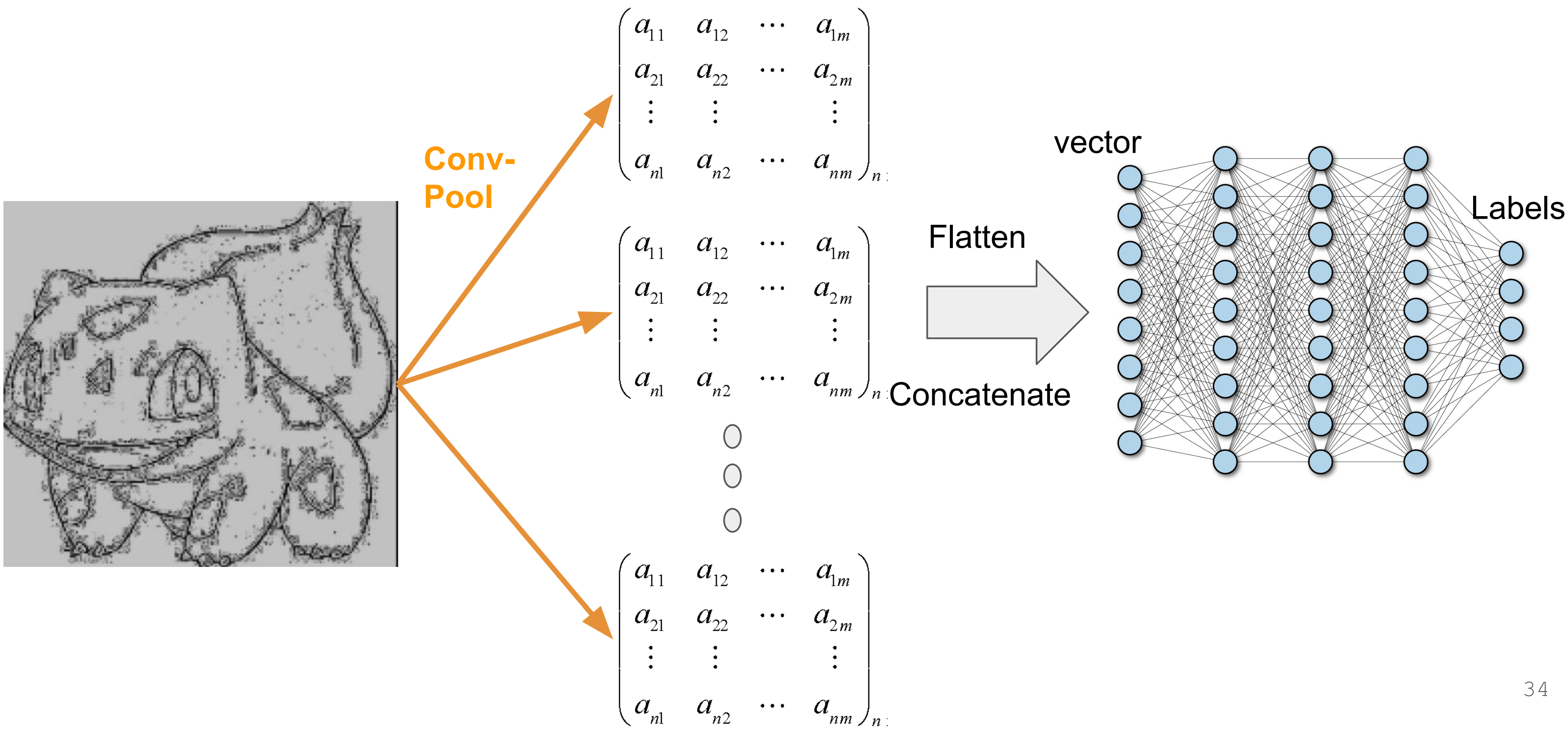
1. The size is **one quarter** the original size
2. The **vertical line** features are **enhanced**.

Flattening

- Flattening is converting the data into one-dimensional array for feeding it to the next layer.

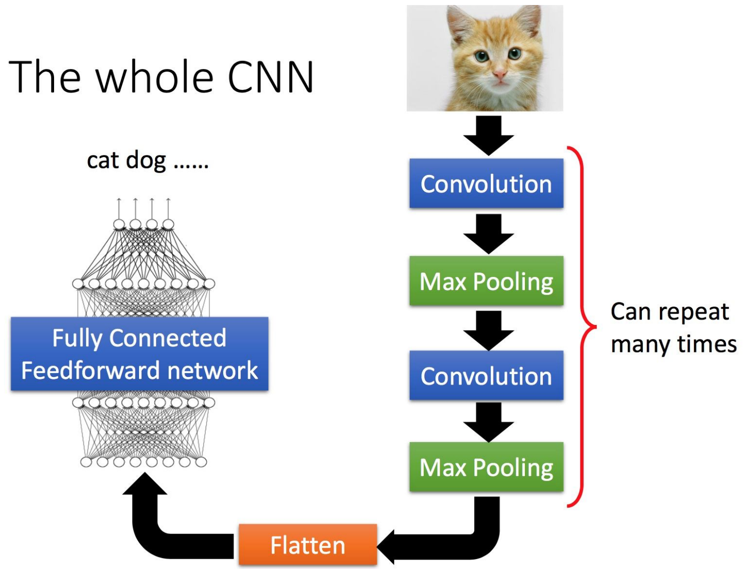


All in one shot



CNN can be deep

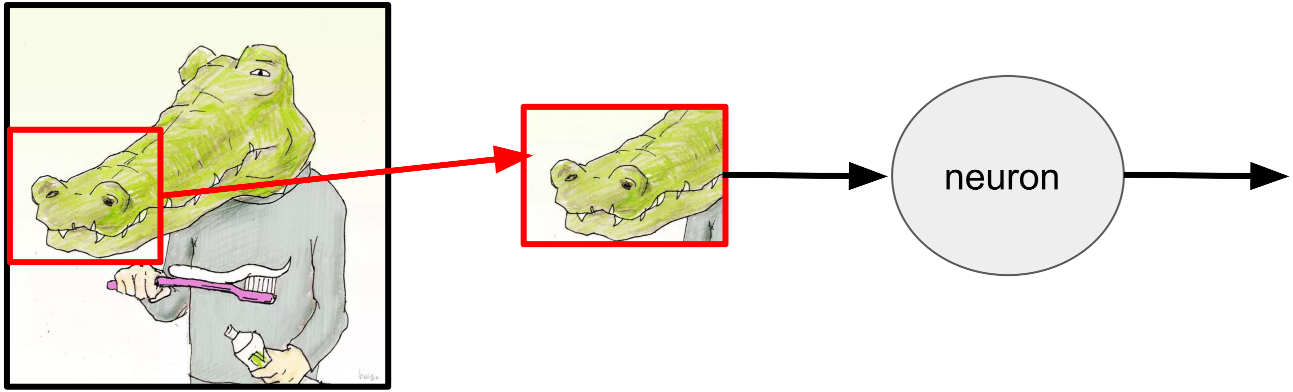
- Conv-Pool can be followed by another Conv-Pool
- At the end, after flatten operation, fully connected layers are used to map the outputs



2. Why CNN for Images

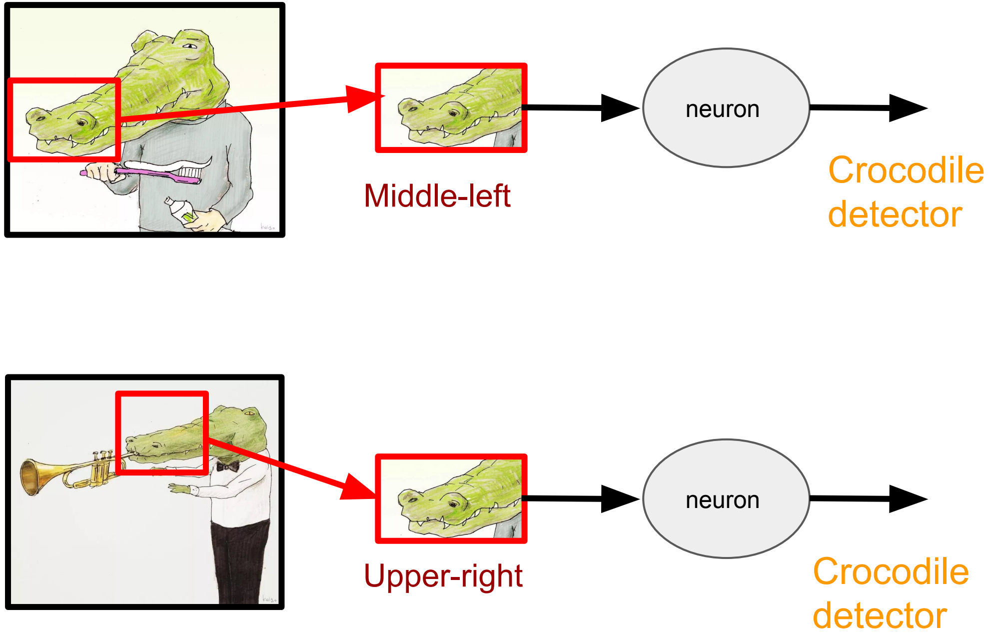
Local features matter

- Discriminative patterns are much smaller than the whole image
- A neuron or feature extractor does not have to see the whole image
- Less parameters required



Location insensitive

- The same patterns appear in different regions
- A neuron should be location insensitive



Subsampling works

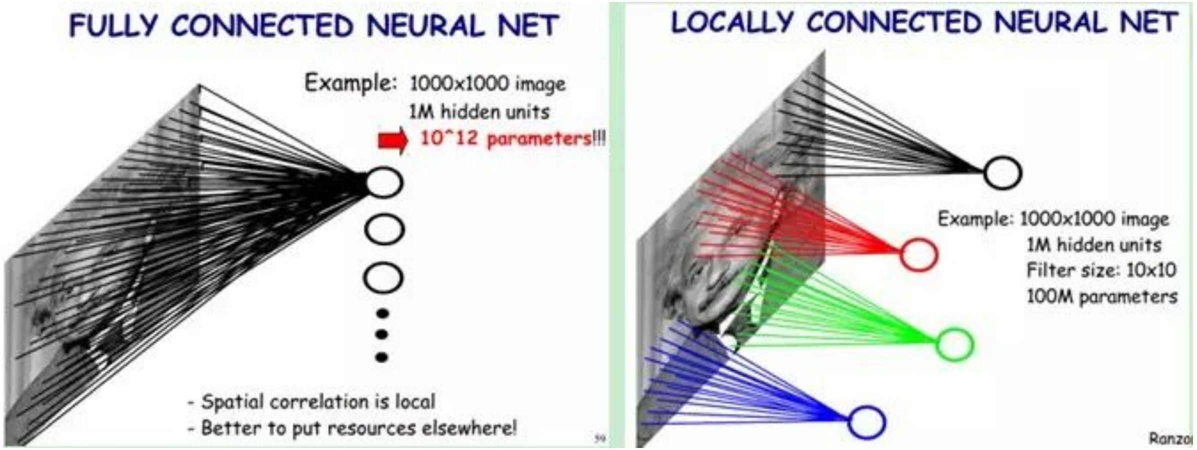
- Subsampling the pixels will not change the object
- We can subsample the pixels to make the images smaller -> less parameters required

Crocodile

subsampling

Crocodile

Locally connected

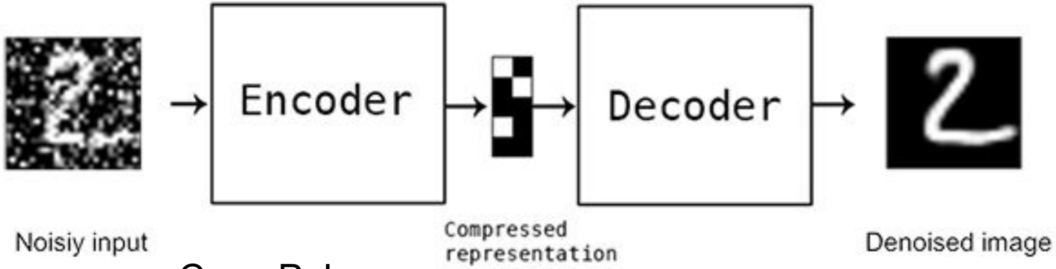


https://cv-tricks.com/cnn/understand-resnet-alexnet-vgg-inception/

Applications

- Image Recognition
- Object Detection
- Image Denoising

Conv-UpSampling2D



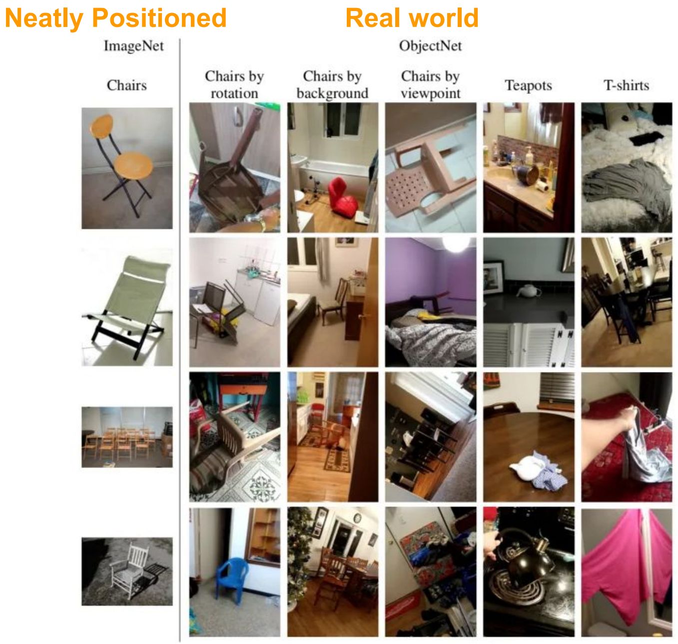
Conv-Pol

https://blog.keras.io/building-autoencoders-in-keras.html
https://www.kaggle.com/michalbrezk/denoise-images-using-autoencoders-tf-keras

3. Limitations of CNN

CNN vs human vision

- CNN can handle translations. But they can not cope with the effects of **changing viewpoints such as rotation and scaling**.
- Huam is able to generalize knowledge.

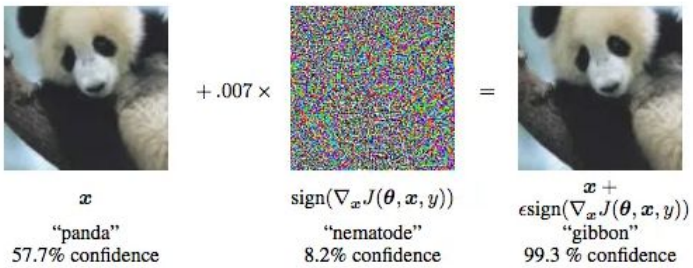


From: objectnet.dev

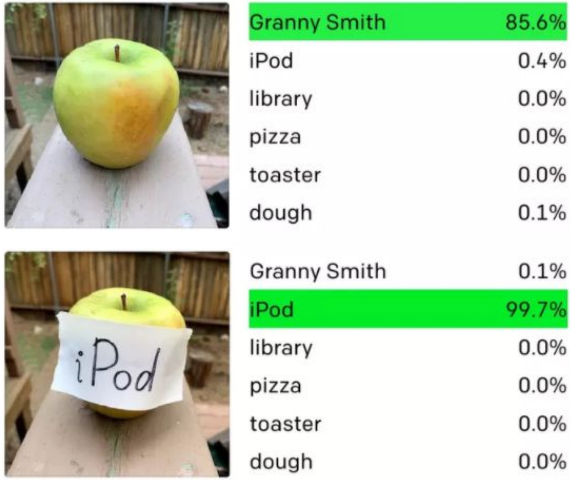
CNN vs human vision

- CNN may get confused by seeing this bizarre teapot, since they can not understand images in terms of objects and their parts.
- Huam is able to decompose an object into parts and then we can understand its nature.

CNN vs human vision



Adversarial examples can cause neural networks to misclassify images while appearing unchanged to the human eye



https://www.theverge.com/2021/3/8/22319173/openai-machine-vision-adversarial-typographic-attacka-clip-multimodal-neuron

Case study

https://medium.com/@DataStevenson/teaching-a-computer-to-classify-anime-8c77bc89b881

Task definition

Training Data

Digimon

Pokemon

Testing Data

Digimon or Pokemon?

Task definition

```
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D
from keras.layers import Activation, Dropout, Flatten, Dense

model = Sequential()
model.add(Conv2D(32, (3, 3), input_shape=(150, 150, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Flatten())  # this converts our 3D feature maps to 1D feature vectors
model.add(Dense(64))
model.add(Activation('relu'))
model.add(Dropout(0.5))
model.add(Dense(1, activation='sigmoid', name='preds'))

model.compile(loss='binary_crossentropy',
              optimizer='rmsprop',
              metrics=['accuracy'])
```

```
Epoch 1/3
8/8 [==============================] - 12s 2s/step - loss: 2.7443 - accuracy: 0.7675 - val_
loss: 0.0834 - val_accuracy: 0.9922
Epoch 2/3
8/8 [==============================] - 12s 2s/step - loss: 0.0560 - accuracy: 0.9835 - val_
loss: 0.0692 - val_accuracy: 0.9961
Epoch 3/3
8/8 [==============================] - 12s 1s/step - loss: 0.0559 - accuracy: 0.9856 - val_
loss: 0.0684 - val_accuracy: 0.9961
```

The implementation and dataset could be found on Canvas Folder-Pokemon vs Digimon

Only after three epochs, the testing/val accuracy was easily over 99%. Amazing!

Next Class: Interpretability Methods in Machine Learning