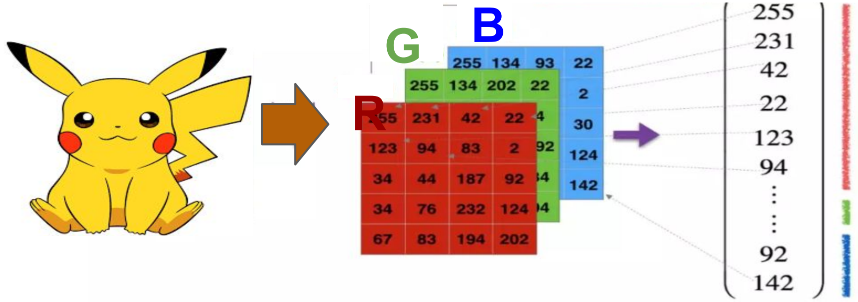
Convolutional Neural Network

Before CNN

Computers See Image

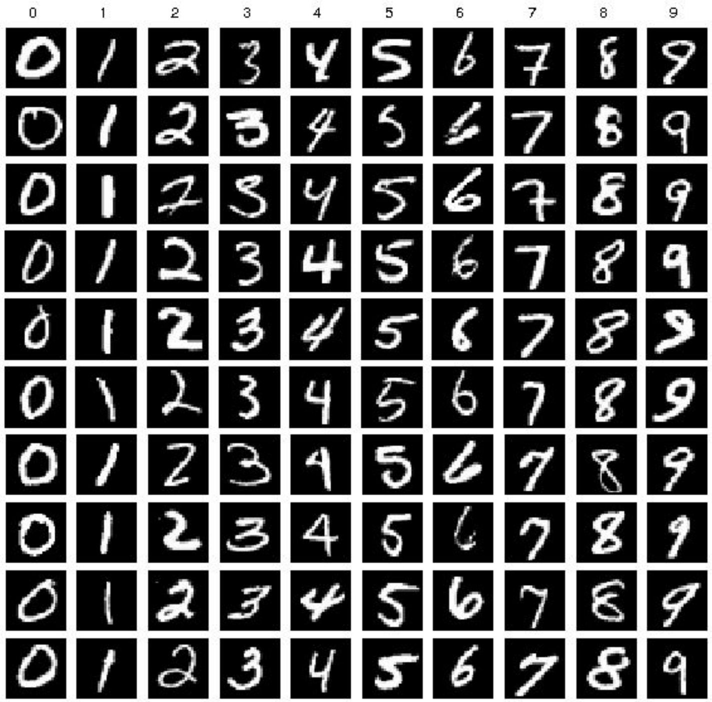


Softmax Classifier
SVM
….

Fully-Connected
Neural Network

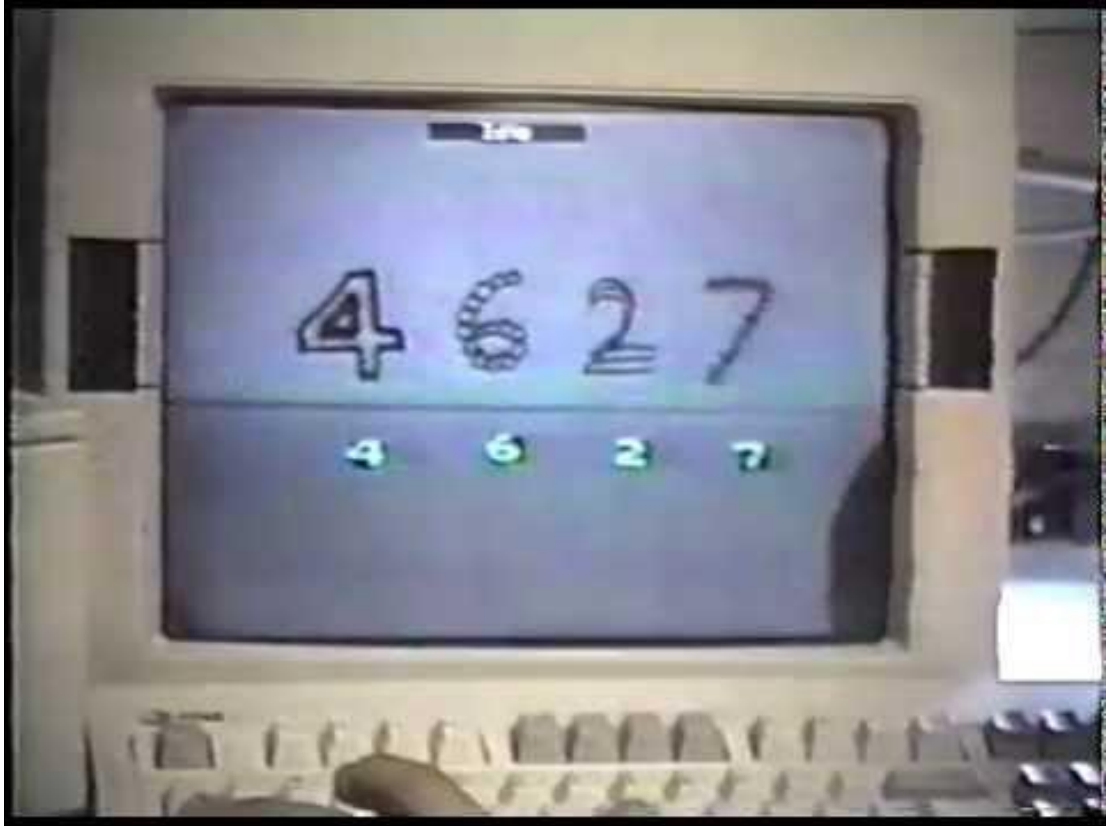
Labels

Think about MNIST Dataset



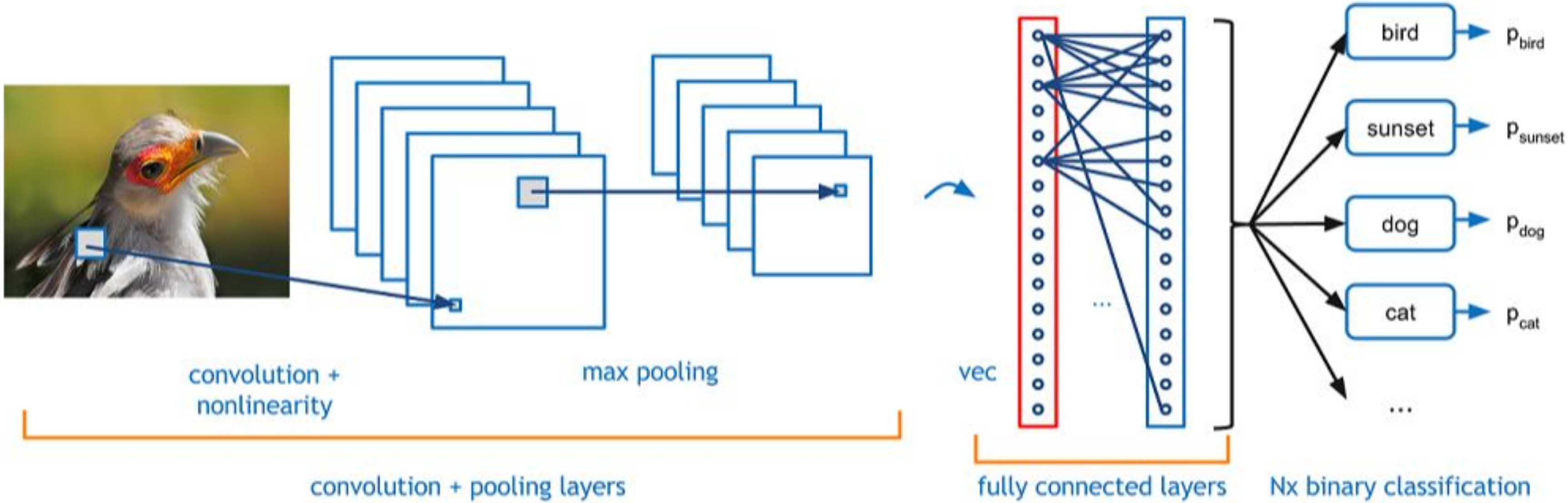
The above model requires the digit should be in the center of the image and it had to be the only thing in the image.

Intro to CNN



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

Convolutional Neural Network

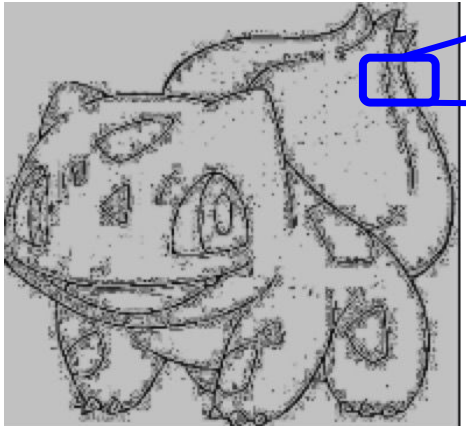


Extracting useful features of data

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

Filter Operation

Consider neighbor values



0	32	35
34	203	122
132	223	163

Current Pixel Value is 203

Dot Product

-1	**0**	**-2**
.5	**4.5**	**-1.5**
1.5	**2**	**-3**

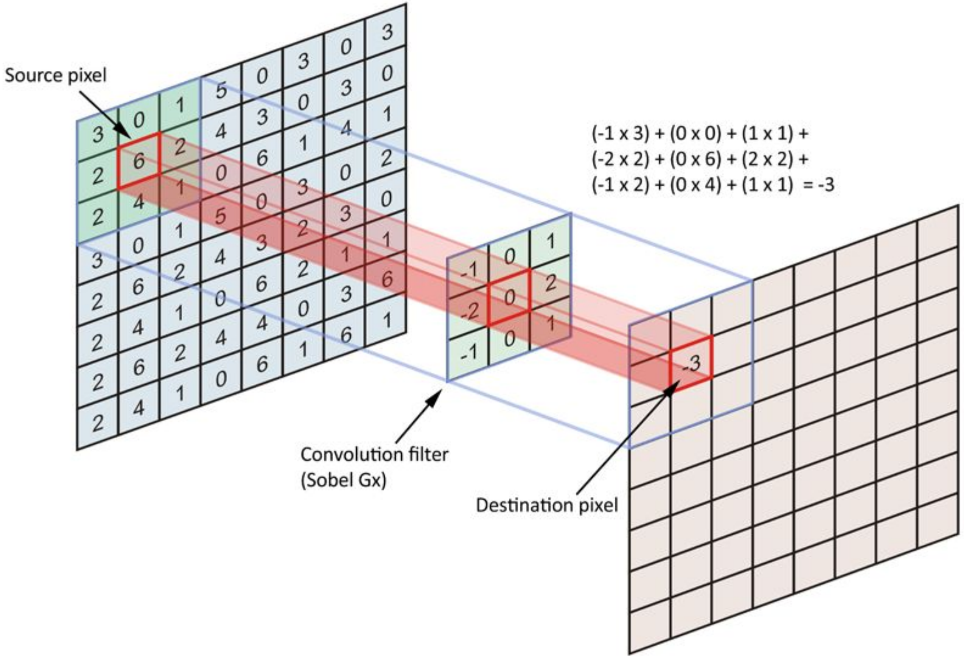
Filter (3 by 3)

New Pixel Value

$$(-1 * 0) + (0 * 32) + (-2 * 35) +$$
$$(.5 * 34) + (4.5 * 203) + (-1.5 * 122) +$$
$$(1.5 * 132) + (2 * 223) + (-3 * 163)$$

Filter Size

Filter Operation

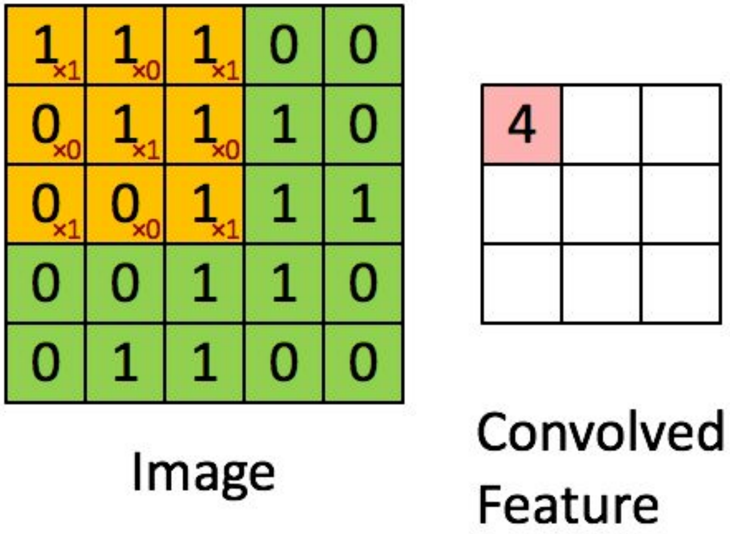


The intent of convolution is to encode source data matrix (entire image) in terms of a filter or kernel. More specifically, we are trying to encode the pixels in the **neighborhood** of **anchor/source** pixels

https://datascience.stackexchange.com/questions/23183/why-convolutions-always-use-odd-numbers-as-filter-size

Convolutional Operation

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



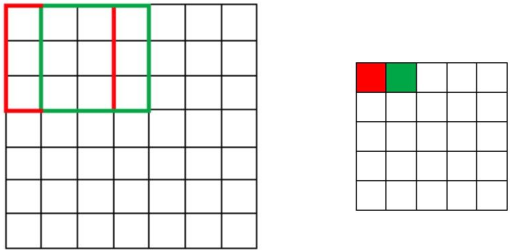
Feature Map

Stanford UFLDL

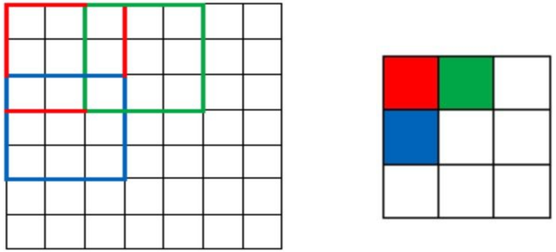
Stride Size

- Controls how the filter move around the image
- It is the amount by which the filter shifts

Stride is 1

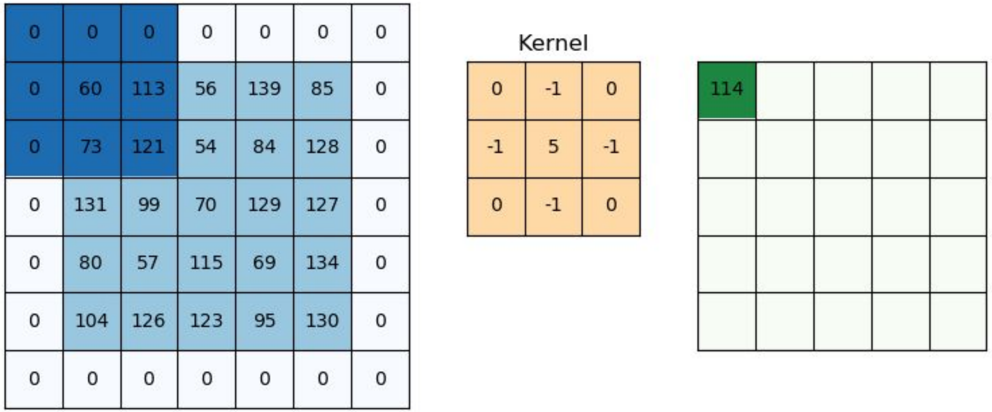


Stride is 2



Padding Size

- Pads the image with zeros around the **border**
- Make the input image and feature map have the same spatial dimensions



Stride: 1
Size of zero padding:
(k-1)/2

https://stackoverflow.com/questions/52067833/how-to-plot-an-animated-matrix-in-matplotlib

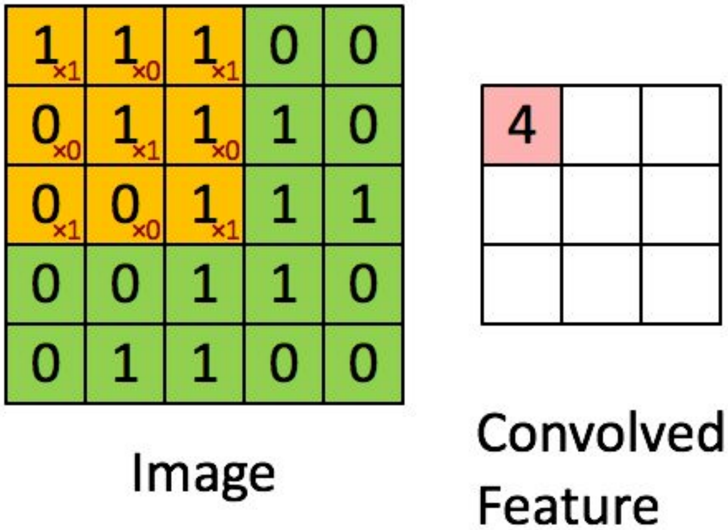
Convolutional Operation

- Filter Size: K
- Stride Size: S
- Padding Size: P

Input size

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

Output size



Stanford UFLDL

Multi-Channel CNN

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

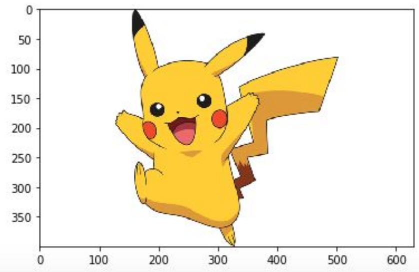
```python
from matplotlib.image import imread
import numpy as np
img = imread('pikka_3.jpg')
```

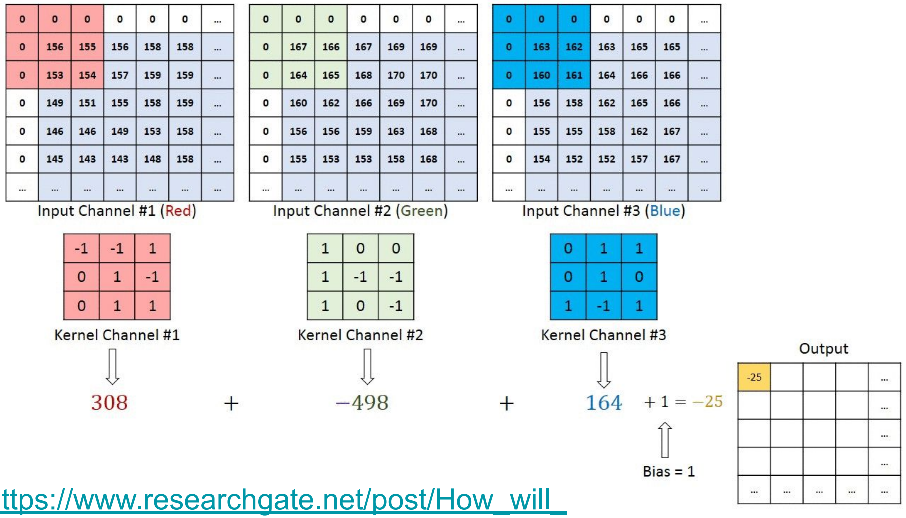
```python
print(img.shape)
```

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

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

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





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

From Keras Layers Conv2D

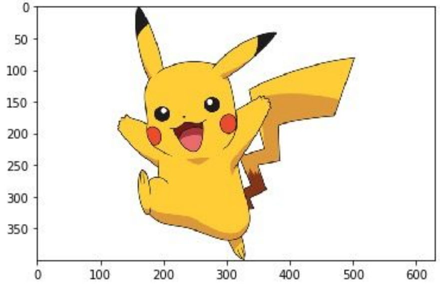
Input shape

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.

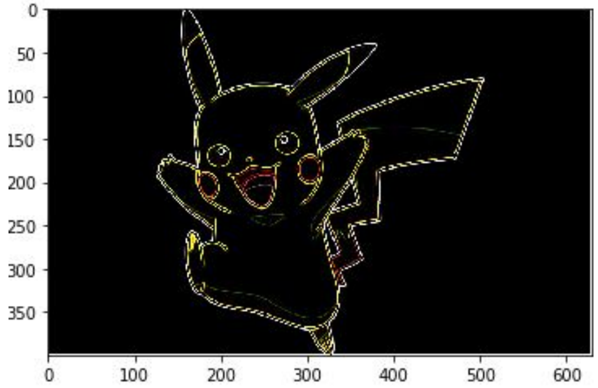
Filter comes from “Image Processing”



Image

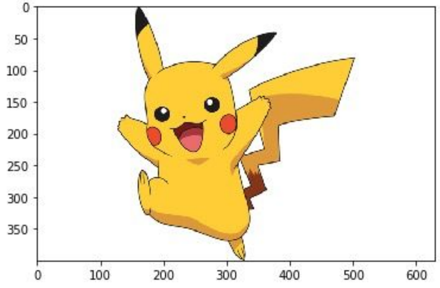
```
print(kernel)

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

Edge Detection



Convolved Features

Filter comes from "Image Processing"

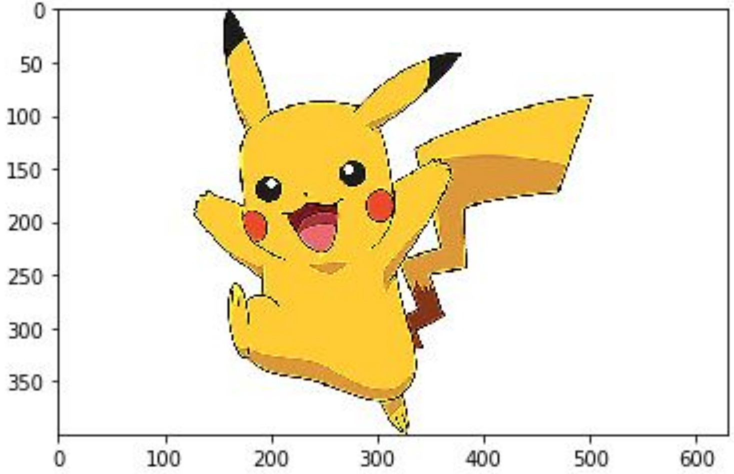


Image

```
print(kernel)
```

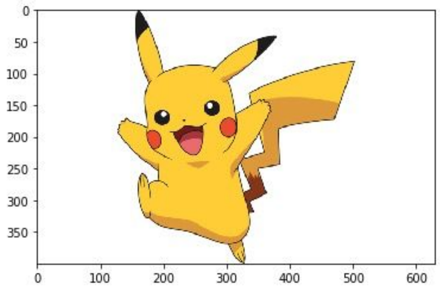
```
[[ 0 -1  0]
 [-1  5 -1]
 [ 0 -1  0]]
```

Sharpen



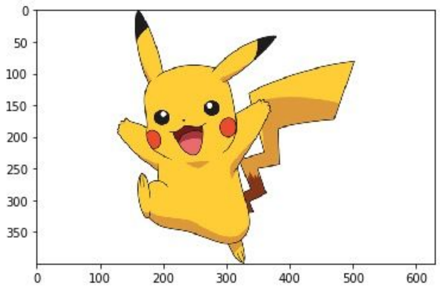
Convolved Features

Filter comes from "Image Processing"



Image

Identity



Convolved Features

Where are these filters from?

- Filters, in nature, are model parameters, which can be **learned** by Gradient Descent Algorithms .

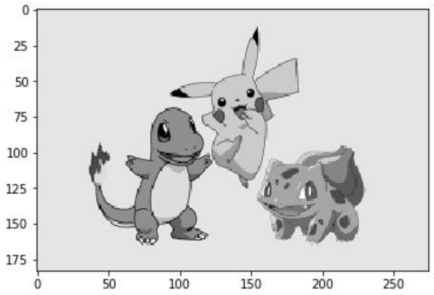
- These filters weights are firstly randomly initialized, and then updated during training process.

- End-to-End optimization: Gradients computed by backpropagation.

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

Non-linear Activation

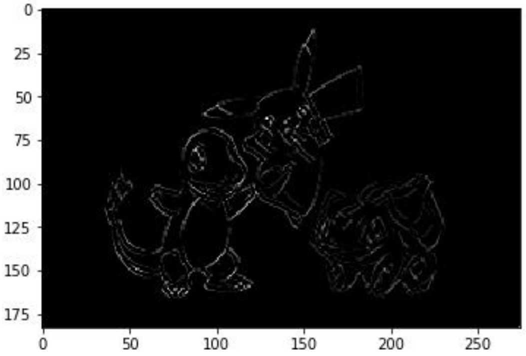
- Filter operation is dot product (linear computation).
- In deep learning, we need to have non-linear transformations.
- Add non-linear activation



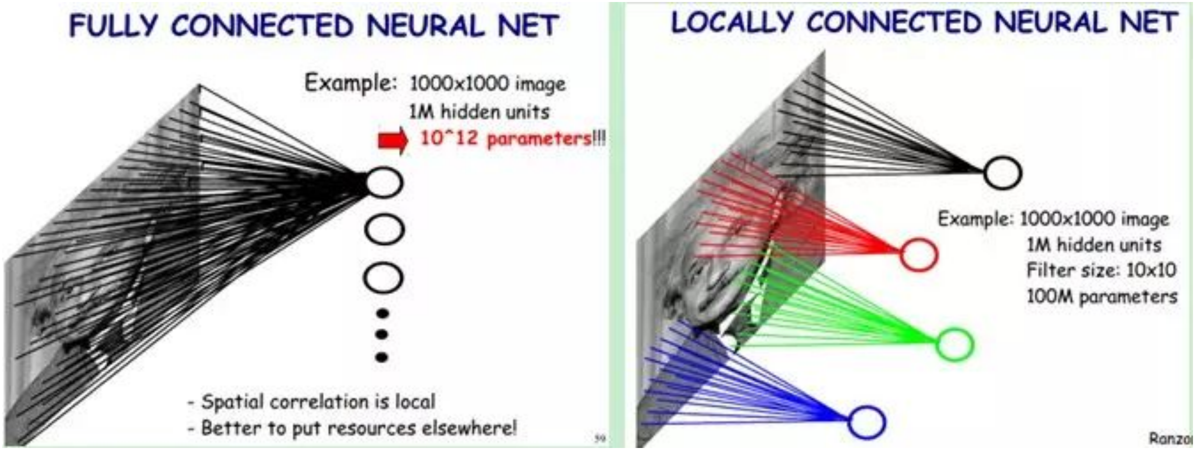
Image

```
print(kernel)

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

non-linear



Locally Connected

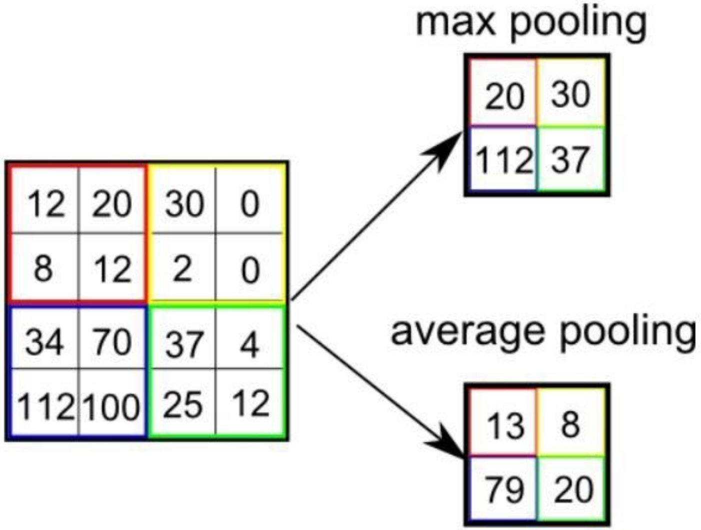


https://cv-tricks.com/cnn/understand-resnet-alexnet-vgg-inception/

Pooling Operation

- Pooling Size: the box size. Here is 2 * 2
- Stride Size: how much pixel the window move
- Reduce the dimensionality

What is stride size here？



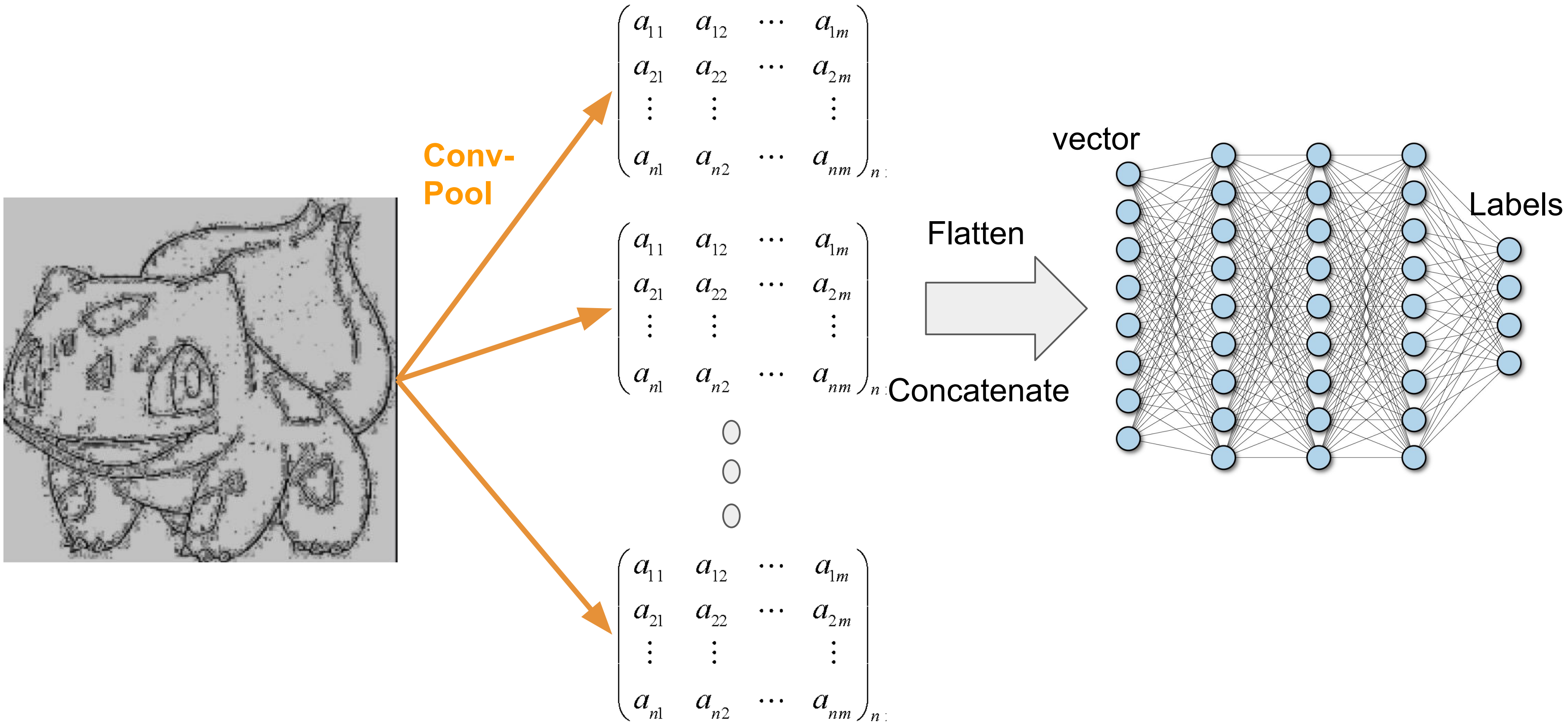
Filter then Pool

-1	0	1
-2	0	2
-1	0	1

2 * 2
max pooling

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

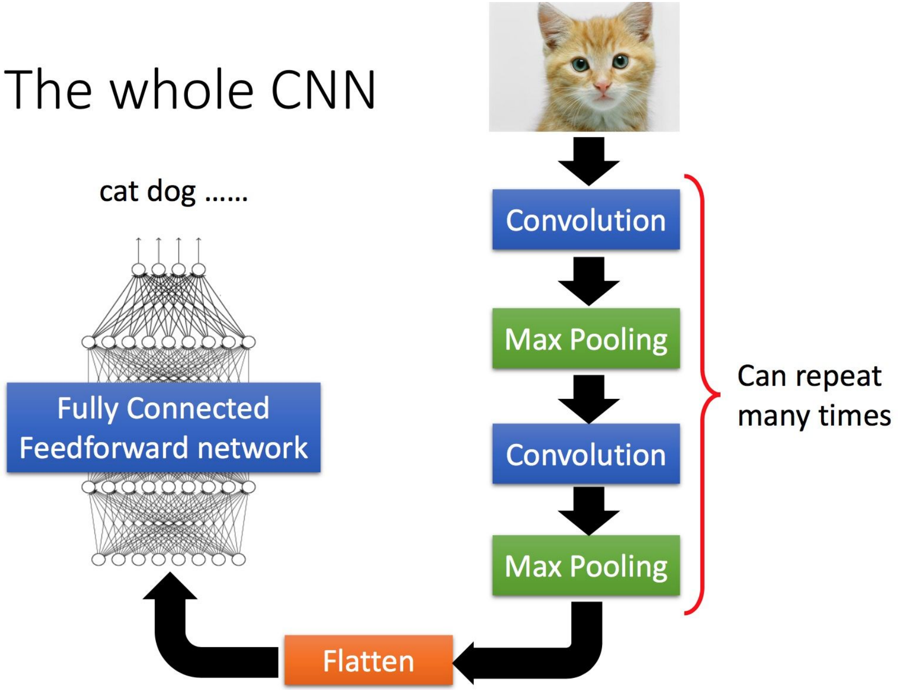
Conv-Pool



CNN Can be Deep

- Convolution-Pooling can be followed by another Convolution-Pooling

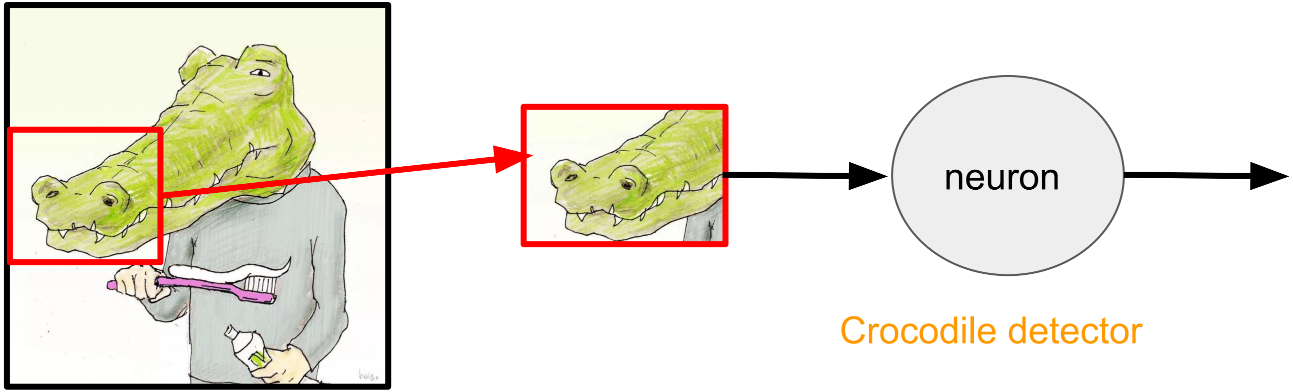
- At the end, after flatten operation, fully connected layers are used to map the outputs.



Why CNN is Suitable for Images

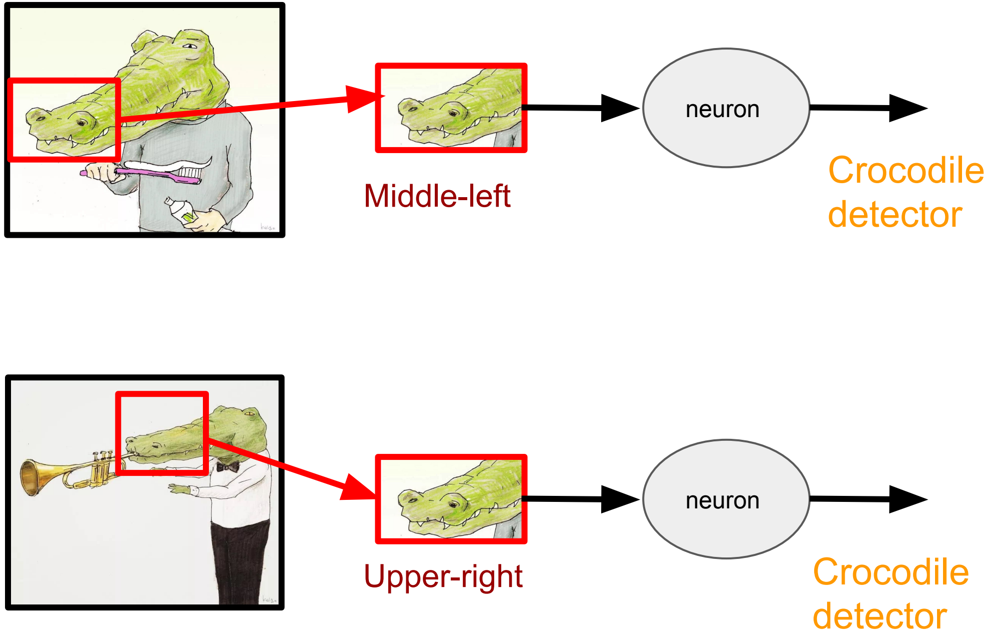
Local Features Matter

- Discriminative patterns are much smaller than the whole image
- A neuron 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 images smaller -> less parameters required

Crocodile

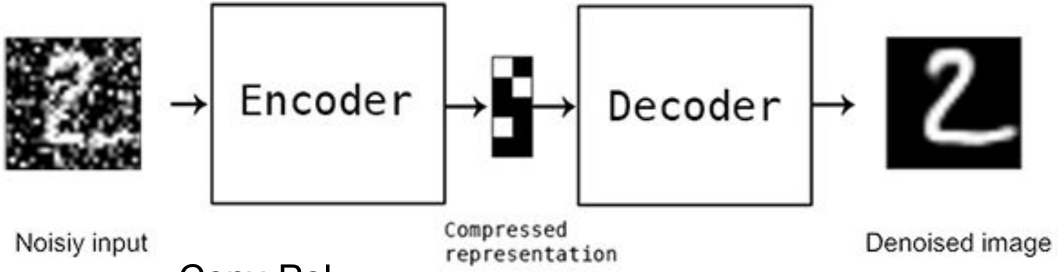
subsampling

Crocodile

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

Limitations of CNN

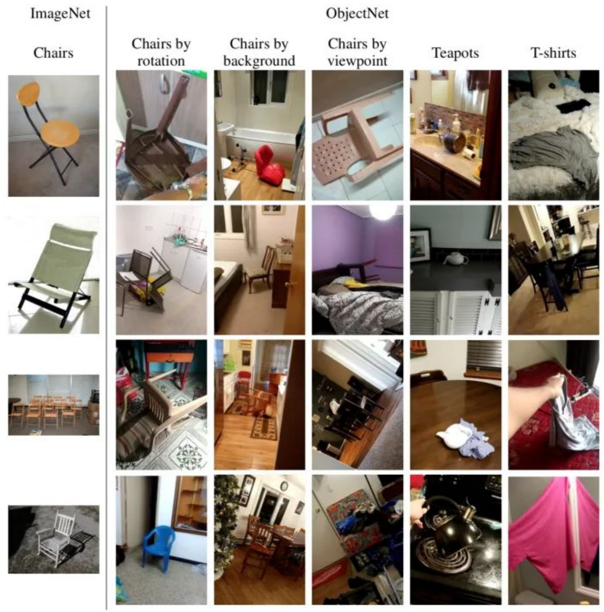
CNN is different human vision

- CNN can handle translations. But they can not cope with the effects of **changing viewpoints such as rotation and scaling**

- Human is able to generalize knowledge.

neatly positioned

Real world



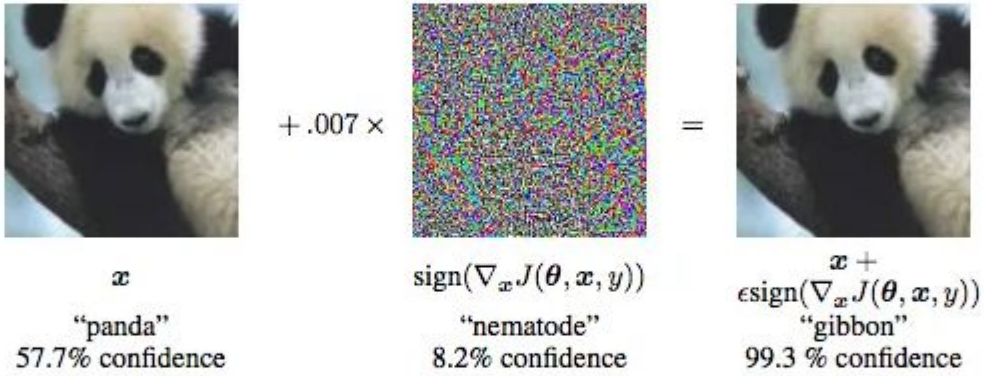
From: objectnet.dev

CNN is different human vision

- CNN may get confused by seeing this bizarre teapot, since they can not understand images in terms of objects and their parts.

- Human is able to decompose an object into parts and then we can understand its nature.

CNN is different human vision



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

CNN for Structured Data

Default of Credit Card Clients Dataset

- Static Features
- **Dynamic Features**

Task: Predict the probability of credit default based on credit card owner's payment status, balance and payment history (for the past 6 months from the predicted period)

Content

There are 25 variables:

- ID: ID of each client
- LIMIT_BAL: Amount of given credit in NT dollars (includes individual and family/supplementary credit
- SEX: Gender (1=male, 2=female)
- EDUCATION: (1=graduate school, 2=university, 3=high school, 4=others, 5=unknown, 6=unknown)
- MARRIAGE: Marital status (1=married, 2=single, 3=others)
- AGE: Age in years
- PAY_0: Repayment status in September, 2005 (-1=pay duly, 1=payment delay for one month, 2=payment delay for two months, ... 8=payment delay for eight months, 9=payment delay for nine months and above)
- PAY_2: Repayment status in August, 2005 (scale same as above)
- PAY_3: Repayment status in July, 2005 (scale same as above)
- PAY_4: Repayment status in June, 2005 (scale same as above)
- PAY_5: Repayment status in May, 2005 (scale same as above)
- PAY_6: Repayment status in April, 2005 (scale same as above)
- BILL_AMT1: Amount of bill statement in September, 2005 (NT dollar)
- BILL_AMT2: Amount of bill statement in August, 2005 (NT dollar)
- BILL_AMT3: Amount of bill statement in July, 2005 (NT dollar)
- BILL_AMT4: Amount of bill statement in June, 2005 (NT dollar)
- BILL_AMT5: Amount of bill statement in May, 2005 (NT dollar)
- BILL_AMT6: Amount of bill statement in April, 2005 (NT dollar)
- PAY_AMT1: Amount of previous payment in September, 2005 (NT dollar)
- PAY_AMT2: Amount of previous payment in August, 2005 (NT dollar)
- PAY_AMT3: Amount of previous payment in July, 2005 (NT dollar)
- PAY_AMT4: Amount of previous payment in June, 2005 (NT dollar)
- PAY_AMT5: Amount of previous payment in May, 2005 (NT dollar)
- PAY_AMT6: Amount of previous payment in April, 2005 (NT dollar)
- default.payment.next.month: Default payment (1=yes, 0=no)

https://www.kaggle.com/uciml/default-of-credit-card-clients-dataset

Feature Engineering

- Extract as much information as possible from the available datasets, especially dynamic features.

- Given the past 6 months bill payments (a sequence of 6 numbers):
    - The averaged bill payment
    - The difference between two consecutive payments
    - …..

Content

There are 25 variables:

- ID: ID of each client
- LIMIT_BAL: Amount of given credit in NT dollars (includes individual and family/supplementary credit
- SEX: Gender (1=male, 2=female)
- EDUCATION: (1=graduate school, 2=university, 3=high school, 4=others, 5=unknown, 6=unknown)
- MARRIAGE: Marital status (1=married, 2=single, 3=others)
- AGE: Age in years
- PAY_0: Repayment status in September, 2005 (-1=pay duly, 1=payment delay for one month, 2=payment delay for two months, … 8=payment delay for eight months, 9=payment delay for nine months and above)
- PAY_2: Repayment status in August, 2005 (scale same as above)
- PAY_3: Repayment status in July, 2005 (scale same as above)
- PAY_4: Repayment status in June, 2005 (scale same as above)
- PAY_5: Repayment status in May, 2005 (scale same as above)
- PAY_6: Repayment status in April, 2005 (scale same as above)
- BILL_AMT1: Amount of bill statement in September, 2005 (NT dollar)
- BILL_AMT2: Amount of bill statement in August, 2005 (NT dollar)
- BILL_AMT3: Amount of bill statement in July, 2005 (NT dollar)
- BILL_AMT4: Amount of bill statement in June, 2005 (NT dollar)
- BILL_AMT5: Amount of bill statement in May, 2005 (NT dollar)
- BILL_AMT6: Amount of bill statement in April, 2005 (NT dollar)
- PAY_AMT1: Amount of previous payment in September, 2005 (NT dollar)
- PAY_AMT2: Amount of previous payment in August, 2005 (NT dollar)
- PAY_AMT3: Amount of previous payment in July, 2005 (NT dollar)
- PAY_AMT4: Amount of previous payment in June, 2005 (NT dollar)
- PAY_AMT5: Amount of previous payment in May, 2005 (NT dollar)
- PAY_AMT6: Amount of previous payment in April, 2005 (NT dollar)
- default.payment.next.month: Default payment (1=yes, 0=no)

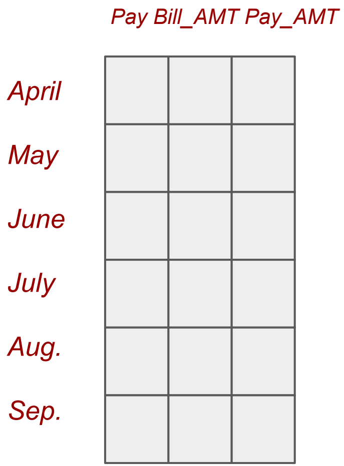
Design of those hand-crafted features is challenging, time-consuming, requires domain knowledge.

Representation of data in CNN format

Shape: 1 by 18

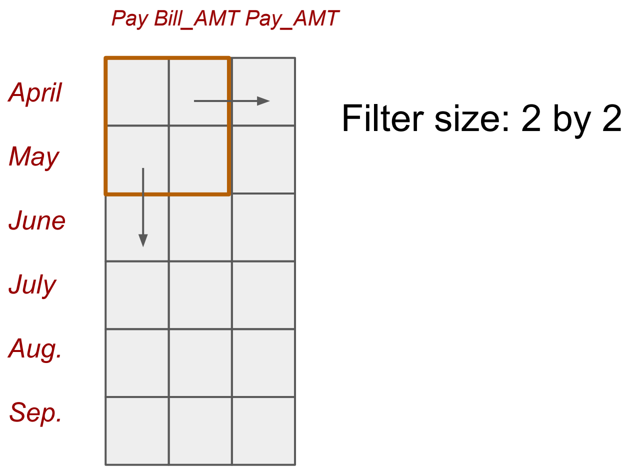
PAY_0
PAY_2
PAY_3
PAY_4
PAY_5
PAY_6
BILL_AMT1
BILL_AMT2
BILL_AMT3
BILL_AMT4
BILL_AMT5
BILL_AMT6
PAY_AMT1
PAY_AMT2
PAY_AMT3
PAY_AMT4
PAY_AMT5
PAY_AMT6

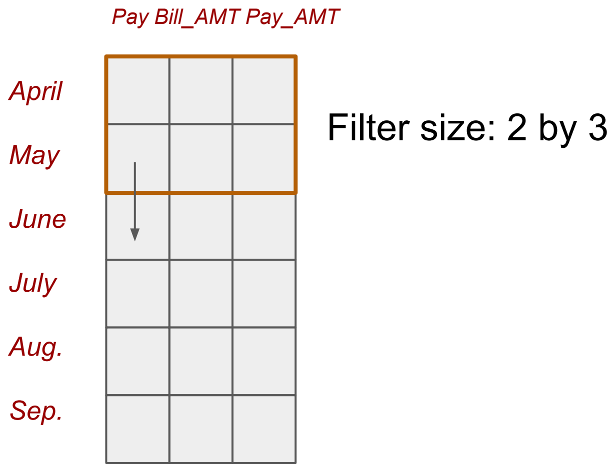
Shape: 1 by 6 by 3



CNN can be easily applied to extract local patterns

Convolution Operation





Which structure is better?

Multiple Channels

- In computer vision, CNN is applied on R-G-B channels
- In this application, different types of credit cards or mortgage of a certain customer can be regarded as different channels

Channel 1

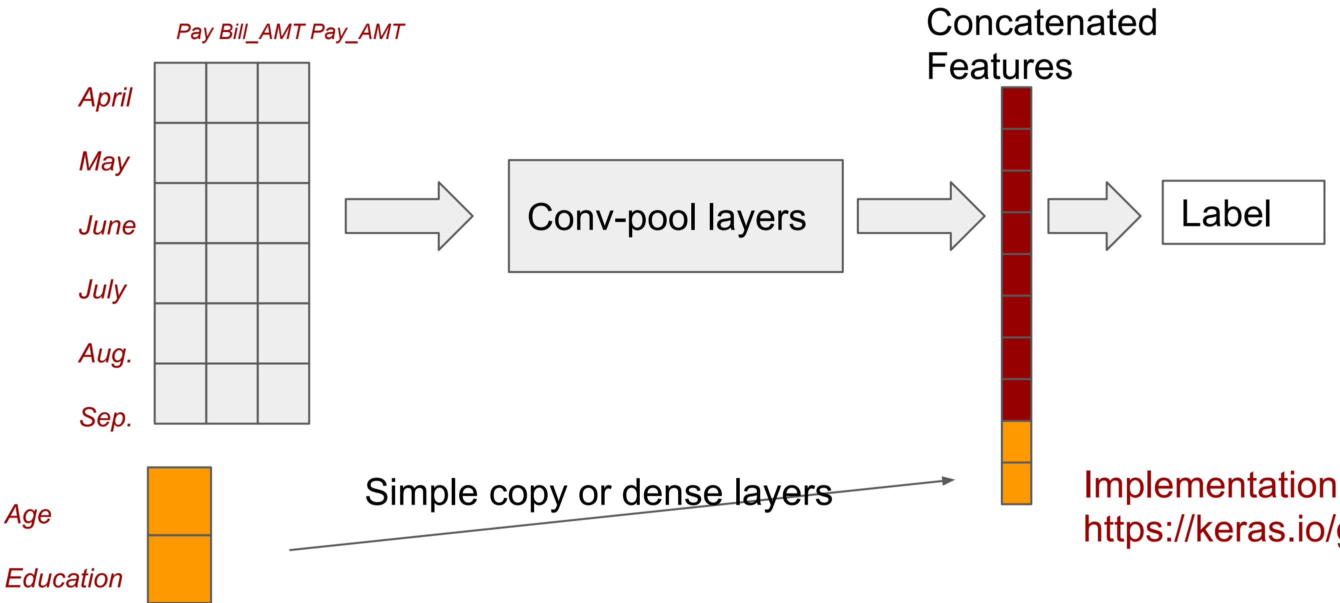
Channel 2

Channel 3

For each customer,
the data shape: 1 by 6 by 3 by **3**

Incorporating Static Features

- Multi-input deep learning is able to combine static and dynamic features for prediction.
- This architecture connects parts of the inputs directly to the output layer.



Implementation Details:
https://keras.io/guides/functional_api/

Can CNN classify digimon and pokemon?

Case Study

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

Task Definition

Training Data

Digimon

Pokemon

Testing Data

Digimon or Pokemon?

Build CNN Model

```
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
```

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