Clasificación de imágenes usando Transfer Learning con Data customizada

Leonardo Leon V.

Outline

- Estado del arte
- Teoría
- Arquitecturas
- Transfer Learning
- Notebook

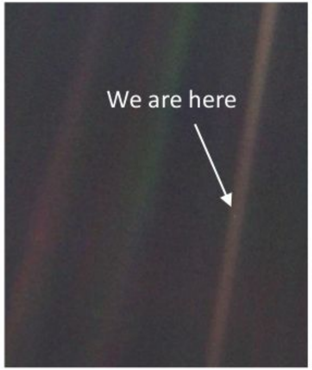
Deep Learning in **One** Slide

- **What is it:**
  Extract useful patterns from data.
- **How:**
  Neural network + optimization
- **How (Practical):**
  Python + TensorFlow & friends
- **Hard Part:**
  Good Questions + Good Data
- **Why now:**
  Data, hardware, community, tools, investment
- **Where do we stand?**
  Most big questions of intelligence have not been answered nor properly formulated

Exciting progress:

- Face recognition
- Image classification
- Speech recognition
- Text-to-speech generation
- Handwriting transcription
- Machine translation
- Medical diagnosis
- Cars: drivable area, lane keeping
- Digital assistants
- Ads, search, social recommendations
- Game playing with deep RL

History of Deep Learning Ideas and Milestones*



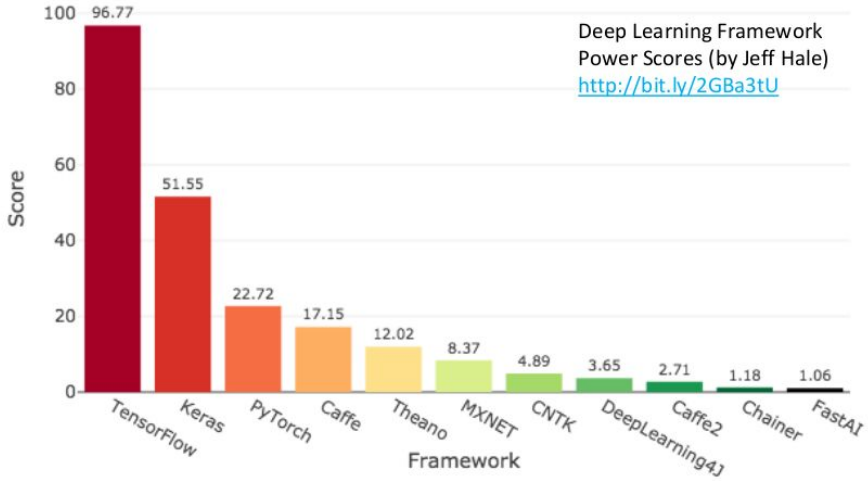
Perspective:

- Universe created
  13.8 billion years ago
- Earth created
  4.54 billion years ago
- Modern humans
  300,000 years ago
- Civilization
  12,000 years ago
- Written record
  5,000 years ago

- 1943: Neural networks
- 1957: Perceptron
- 1974-86: Backpropagation, RBM, RNN
- 1989-98: CNN, MNIST, LSTM, Bidirectional RNN
- 2006: "Deep Learning", DBN
- 2009: ImageNet
- 2012: AlexNet, Dropout
- 2014: GANs
- 2014: DeepFace
- 2016: AlphaGo
- 2017: AlphaZero, Capsule Networks
- 2018: BERT

* Dates are for perspective and not as definitive historical record of invention or credit

Deep Learning Frameworks

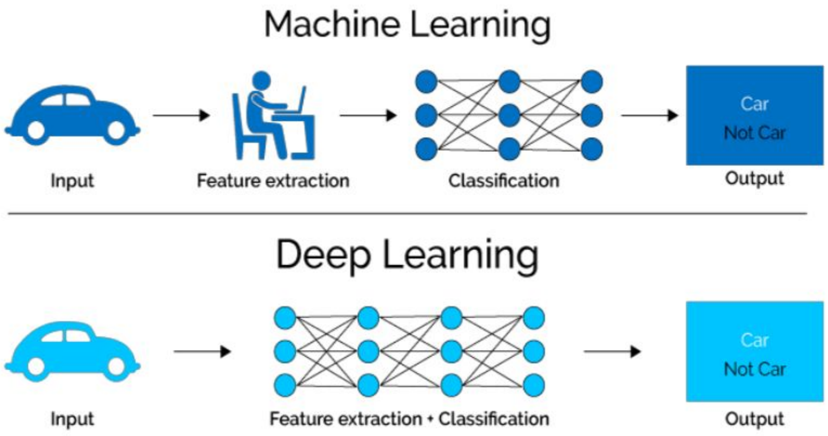


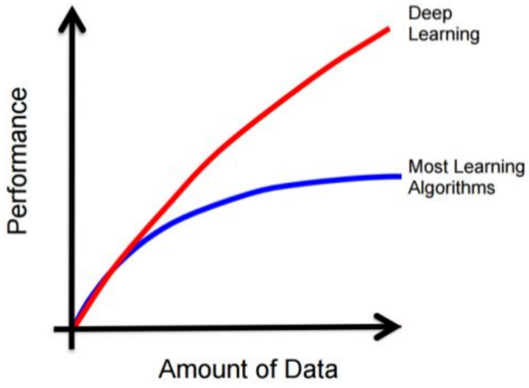
Factors to consider:

- Learning curve
- Speed of development
- Size and passion of community
- Number of papers implemented in framework
- Likelihood of long-term growth and stability
- Ecosystem of tooling

1. TensorFlow
2. Keras
3. PyTorch
4. Caffe
5. theano
6. Apache mxnet
7. Microsoft CNTK
8. DL4J
9. Caffe2
10. Chainer
11. fast.ai

Why Deep Learning? **Scalable** Machine Learning

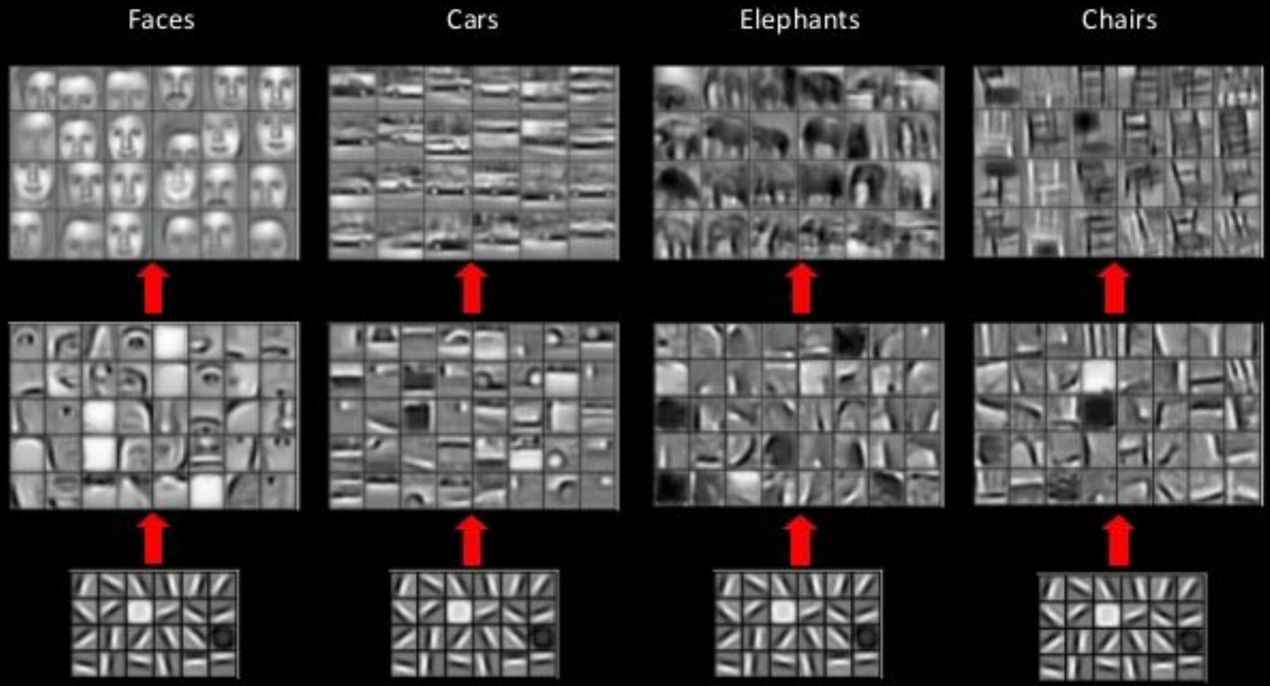


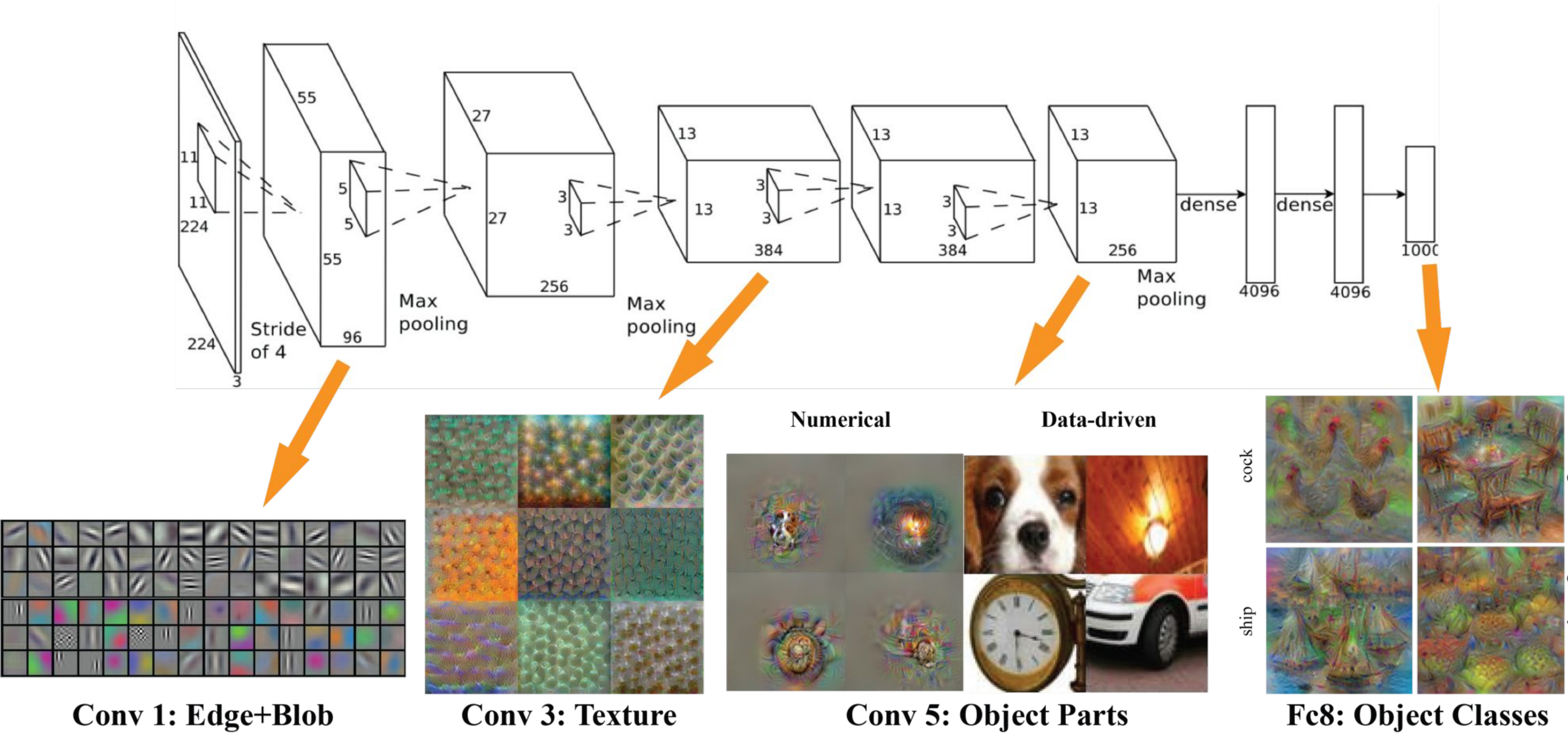


Interpretability of Deep Learning:
https://www.youtube.com/watch?v=93Xv8vJ2acI

Learning of object parts

Examples of learned object parts from object categories





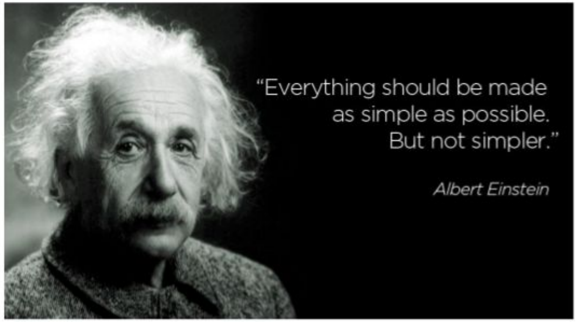
11
11
224
224
3
55
55
96
Stride of 4
5
5
Max pooling
27
27
256
3
3
Max pooling
13
13
384
3
3
13
13
384
3
3
13
13
256
Max pooling
dense
dense
4096
4096
Numerical
Data-driven
cock
ship
dinning table
grocery store
Conv 1: Edge+Blob
Conv 3: Texture
Conv 5: Object Parts
Fc8: Object Classes

The Challenge of Deep Learning:

Efficient Teaching + Efficient Learning

- Humans can learn from very few examples
- Machines (in most cases) need thousands/millions of examples

First Steps: **Start Simple**



Input Image:

TensorFlow Model: Neural Network

Output: 5

(with 87% confidence)

1

```python
# import tensorflow and keras (tf.keras not "vanilla" Keras)
import tensorflow as tf
from tensorflow import keras
```

2

```python
# get data
(train_images, train_labels), (test_images, test_labels) = \
keras.datasets.mnist.load_data()
```

3

```python
# setup model
model = keras.Sequential([
    keras.layers.Flatten(input_shape=(28, 28)),
    keras.layers.Dense(128, activation=tf.nn.relu),
    keras.layers.Dense(10, activation=tf.nn.softmax)
])

model.compile(optimizer=tf.train.AdamOptimizer(),
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
```

4

```python
# train model
model.fit(train_images, train_labels, epochs=5)
```

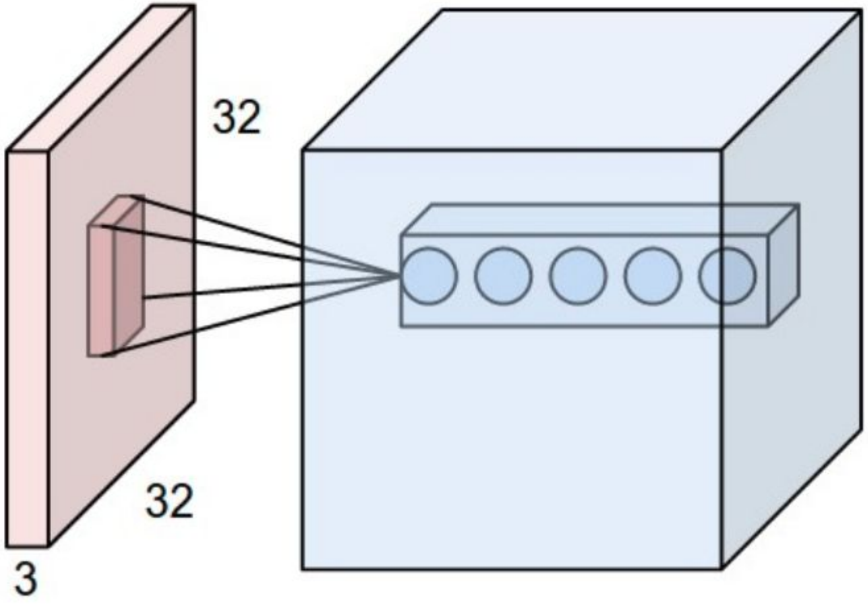
5

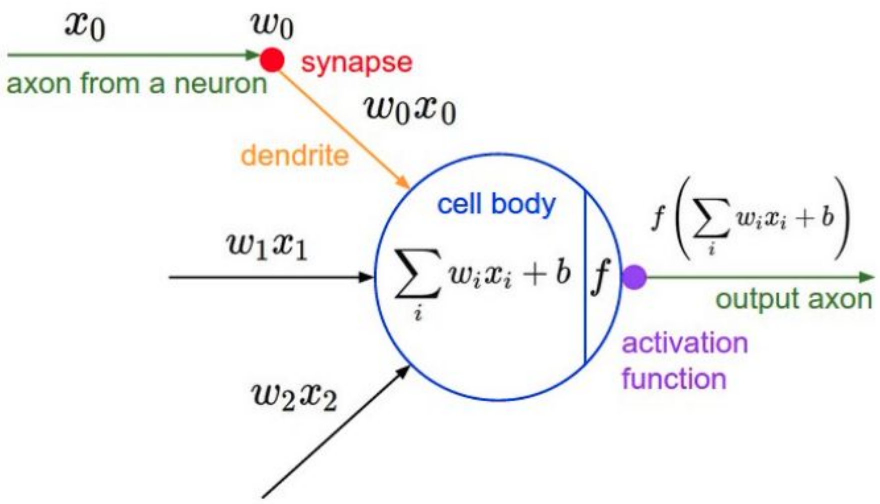
```python
# evaluate
test_loss, test_acc = model.evaluate(test_images, test_labels)
print('test accuracy:', test_acc)
```

6

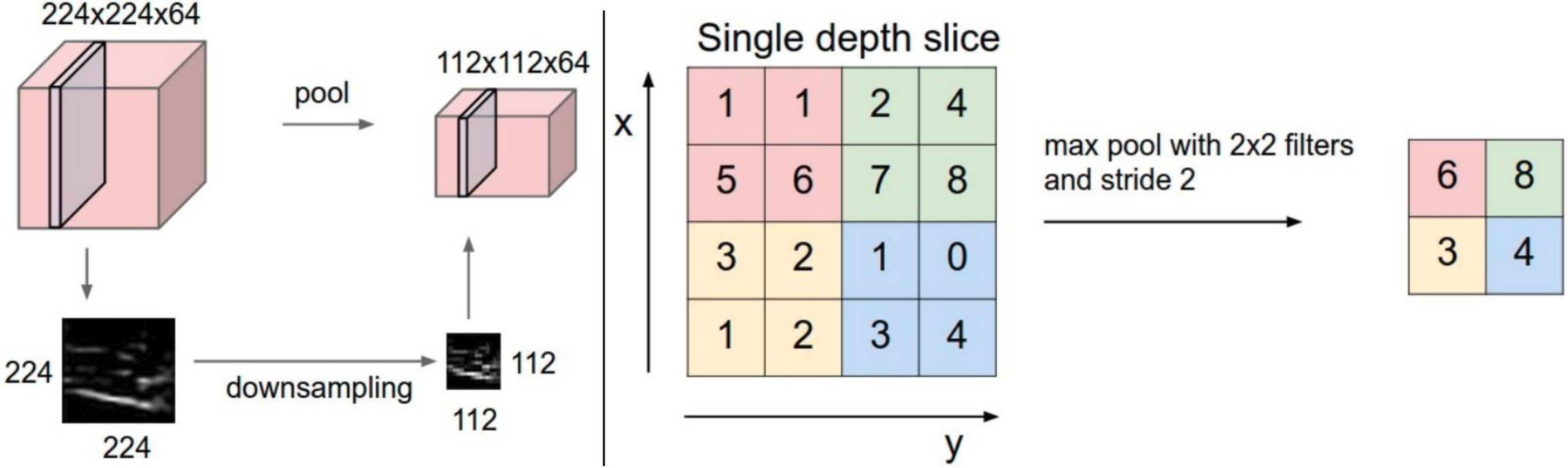
```python
# make predictions
predictions = model.predict(test_images)
```

Convolutional Layer



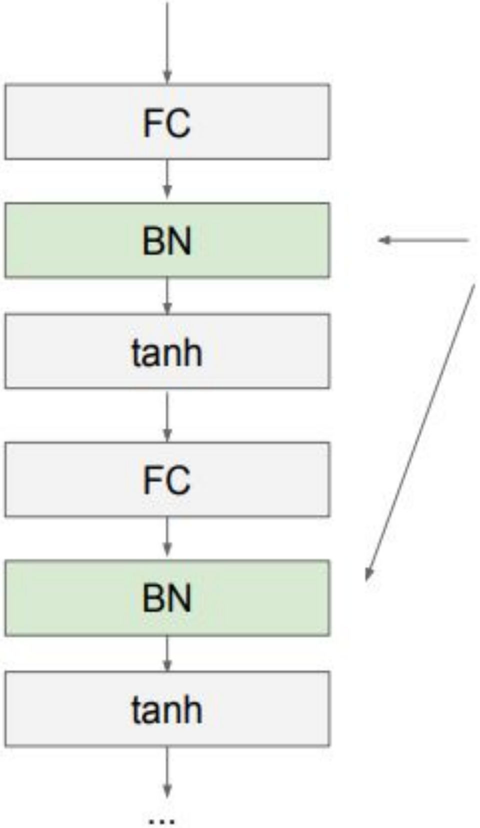


Pooling Layer



Batch Normalization

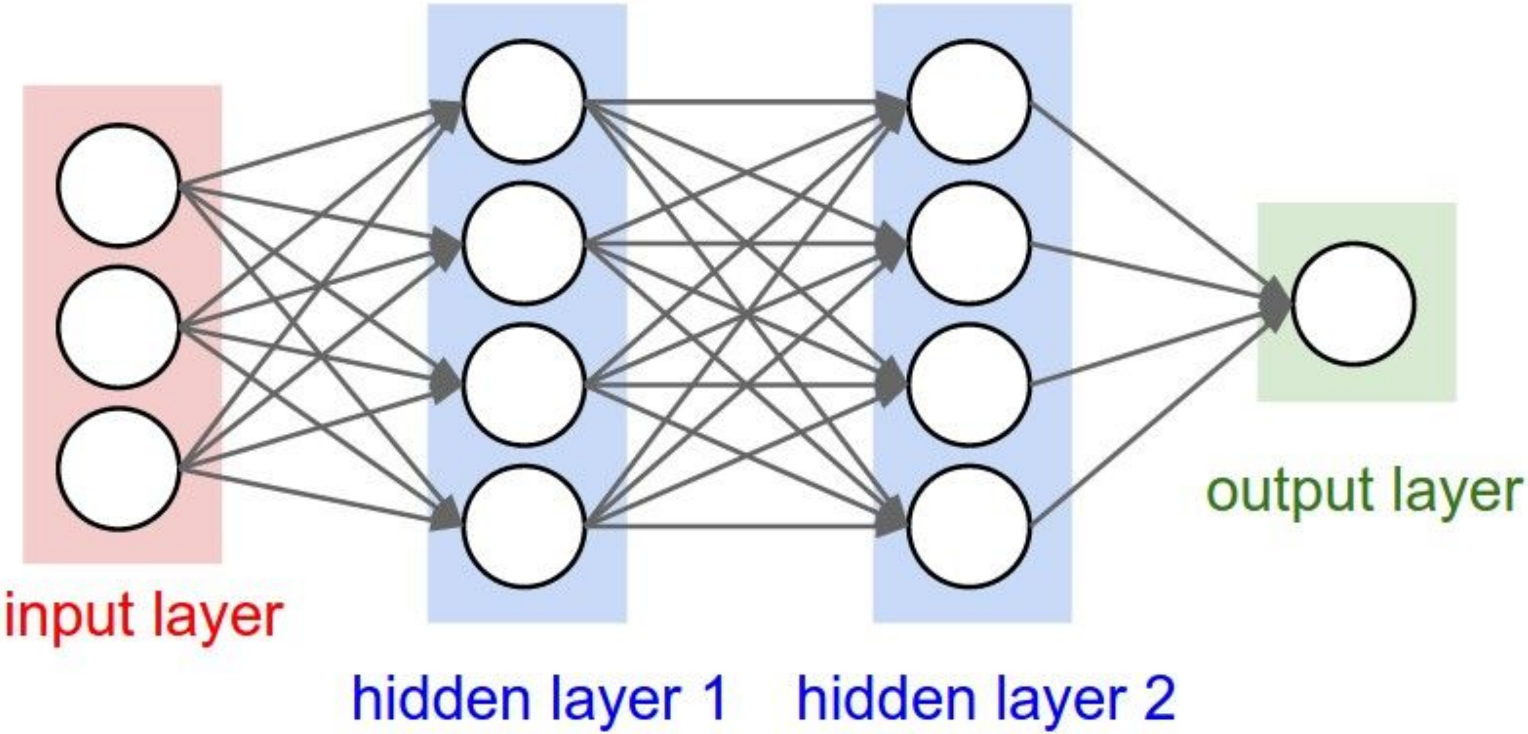
[Ioffe and Szegedy, 2015]



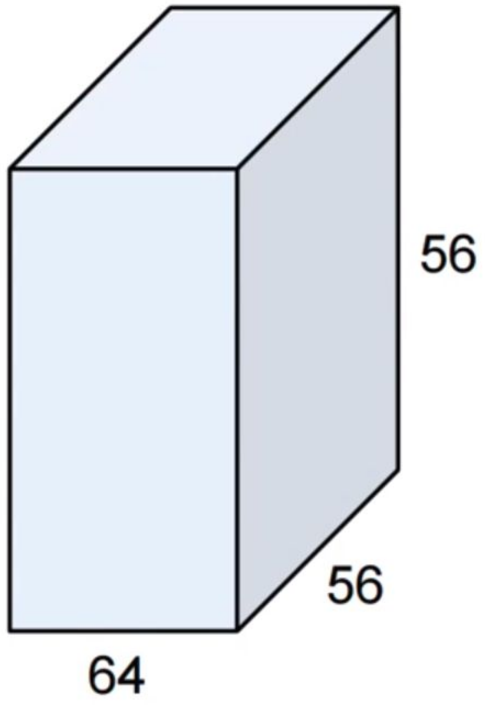
Usually inserted after Fully Connected or Convolutional layers, and before nonlinearity.

$$\widehat{x}^{(k)} = \frac{x^{(k)} - \mathrm{E}[x^{(k)}]}{\sqrt{\mathrm{Var}[x^{(k)}]}}$$

Fully Connected Layer

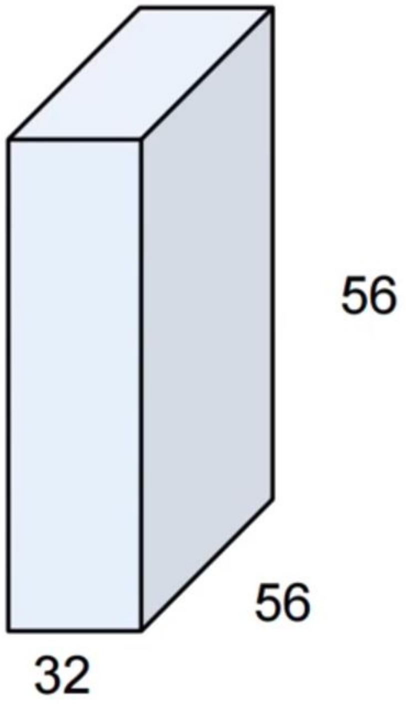


Convolution 1x1



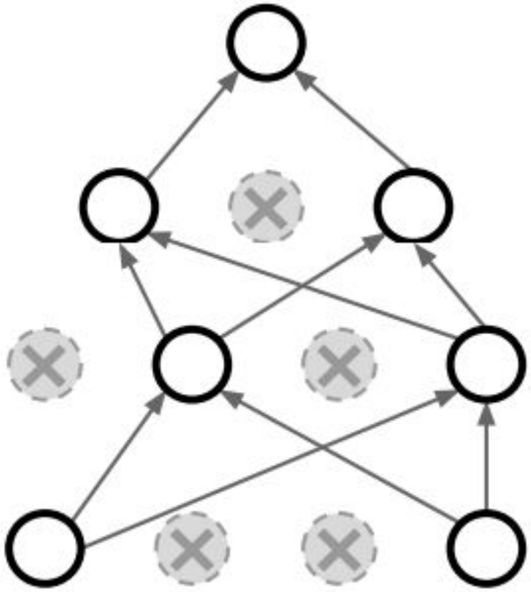
1x1 CONV
with 32 filters

(each filter has size 1x1x64, and performs a 64-dimensional dot product)



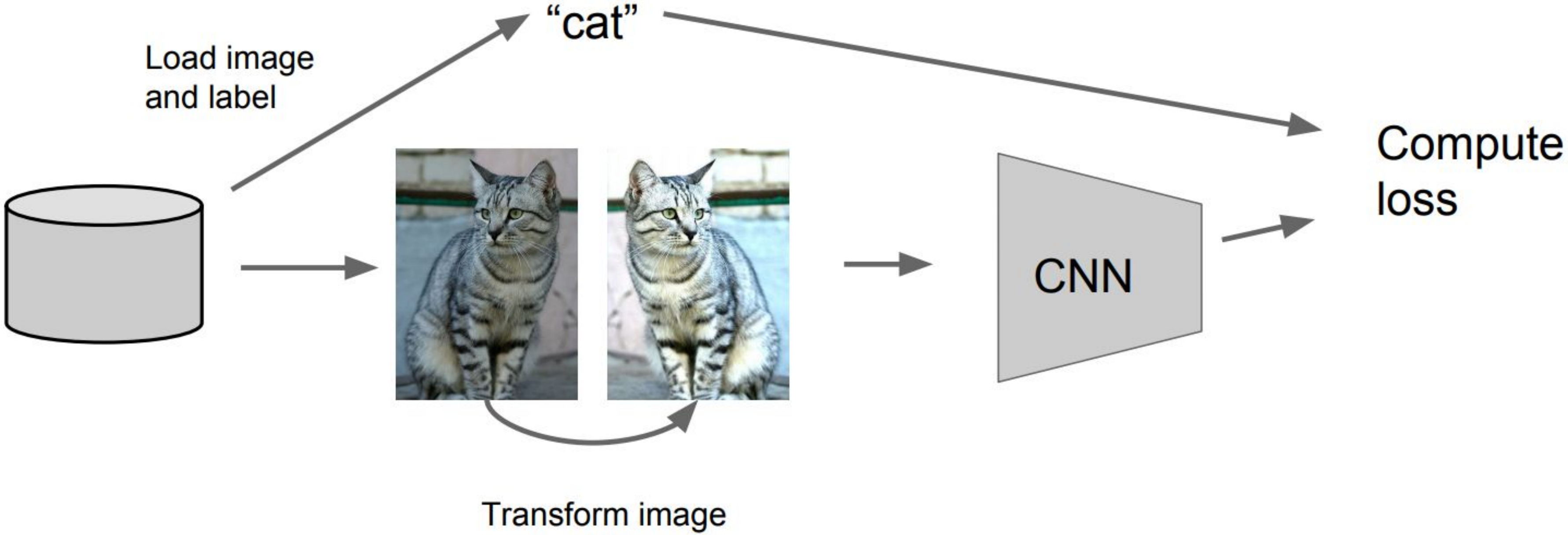
Regularization: Dropout

How can this possibly be a good idea?



Forces the network to have a redundant representation;
Prevents co-adaptation of features

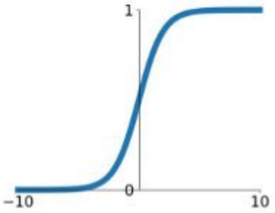
Regularization: Data Augmentation



Activation Functions

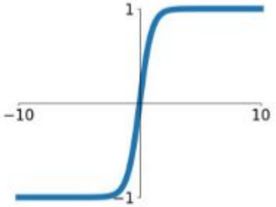
Sigmoid

$\sigma(x) = \frac{1}{1+e^{-x}}$



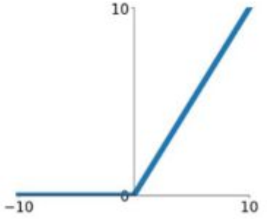
tanh

$\tanh(x)$



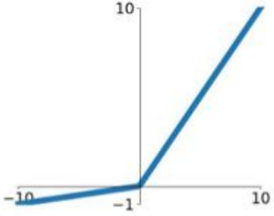
ReLU

$\max(0, x)$



Leaky ReLU

$\max(0.1x, x)$

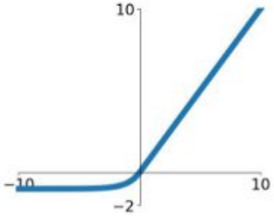


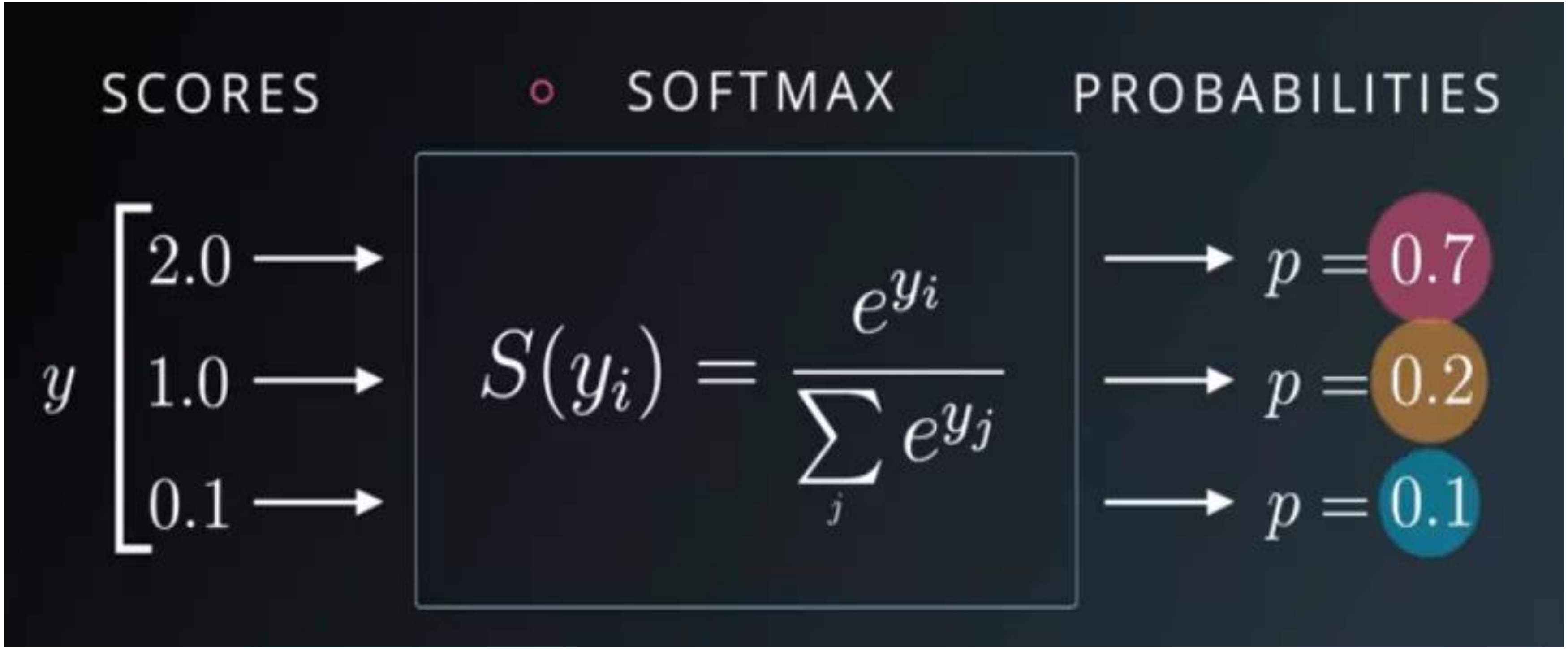
Maxout

$\max(w_1^T x + b_1, w_2^T x + b_2)$

ELU

$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$




SCORES
SOFTMAX
PROBABILITIES
y
2.0
1.0
0.1
$S(y_i) = \frac{e^{y_i}}{\sum_j e^{y_j}}$
p = 0.7
p = 0.2
p = 0.1

CROSS-ENTROPY

$S(Y)$

$$\begin{bmatrix} 0.7 \\ 0.2 \\ 0.1 \end{bmatrix}$$

$$D(S,L) = -\sum_i L_i \log(S_i)$$

L

$$\begin{bmatrix} 1.0 \\ 0.0 \\ 0.0 \end{bmatrix}$$

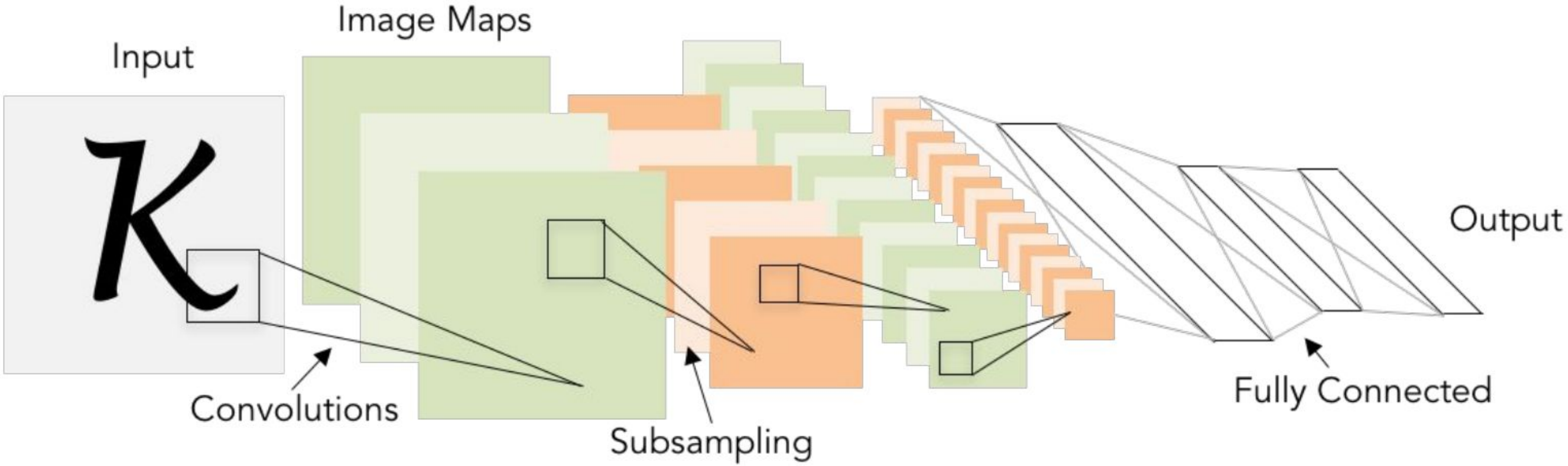
$$D(S,L) \neq D(L,S)$$

Arquitecturas

- LeNet
- AlexNet
- ZFNet
- VGG
- GoogleNet
- ResNet
- InceptionResNet
- NASNet
- Xception
- MobileNet
- SeNet

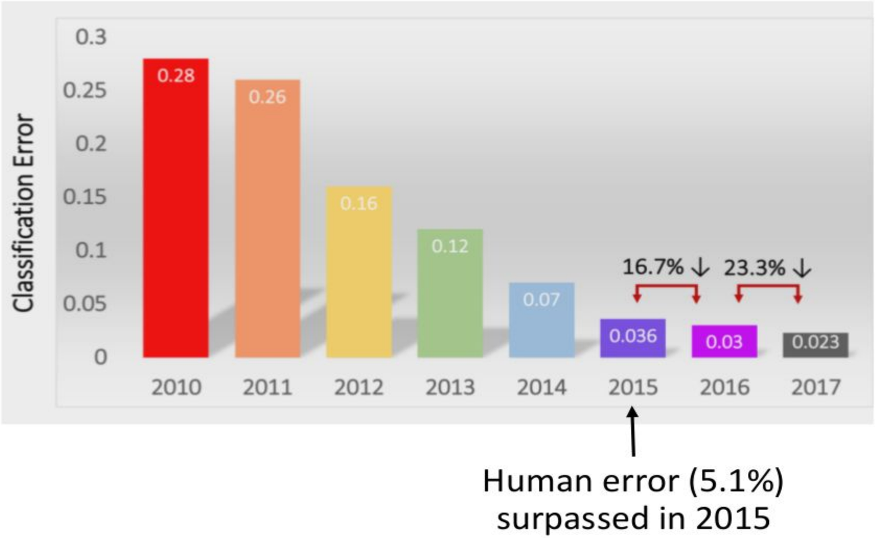
Review: LeNet-5

[LeCun et al., 1998]



Conv filters were 5x5, applied at stride 1
Subsampling (Pooling) layers were 2x2 applied at stride 2
i.e. architecture is [CONV-POOL-CONV-POOL-FC-FC]

Modelos CNN Clasificación

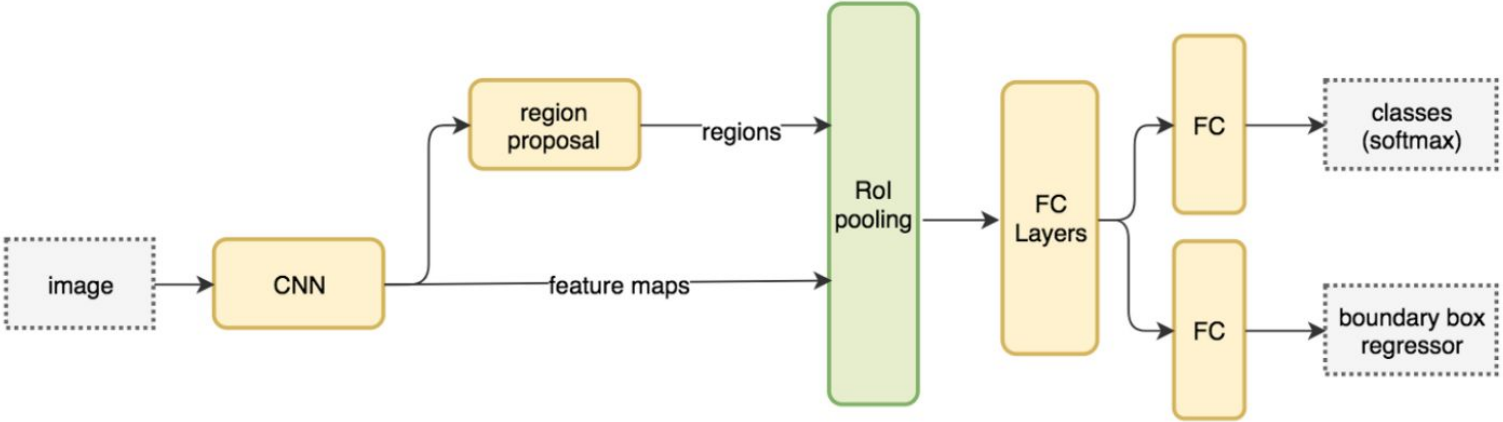


- **AlexNet (2012): First CNN (15.4%)**
  - 8 layers
  - 61 million parameters
- **ZFNet (2013): 15.4% to 11.2%**
  - 8 layers
  - More filters. Denser stride.
- **VGGNet (2014): 11.2% to 7.3%**
  - Beautifully uniform:
    3x3 conv, stride 1, pad 1, 2x2 max pool
  - 16 layers
  - 138 million parameters
- **GoogLeNet (2014): 11.2% to 6.7%**
  - Inception modules
  - 22 layers
  - 5 million parameters
    (throw away fully connected layers)
- **ResNet (2015): 6.7% to 3.57%**
  - More layers = better performance
  - 152 layers
- **CUImage (2016): 3.57% to 2.99%**
  - Ensemble of 6 models
- **SENet (2017): 2.99% to 2.251%**
  - Squeeze and excitation block: network is allowed to adaptively adjust the weighting of each feature map in the convolutional block.

https://keras.io/applications/

Object Detection / Localization

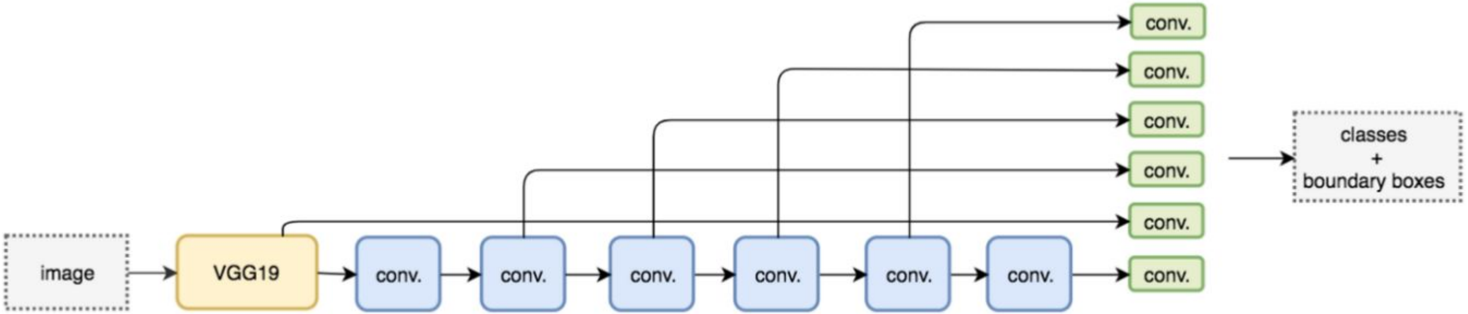
Region-Based Methods | Shown: Faster R-CNN



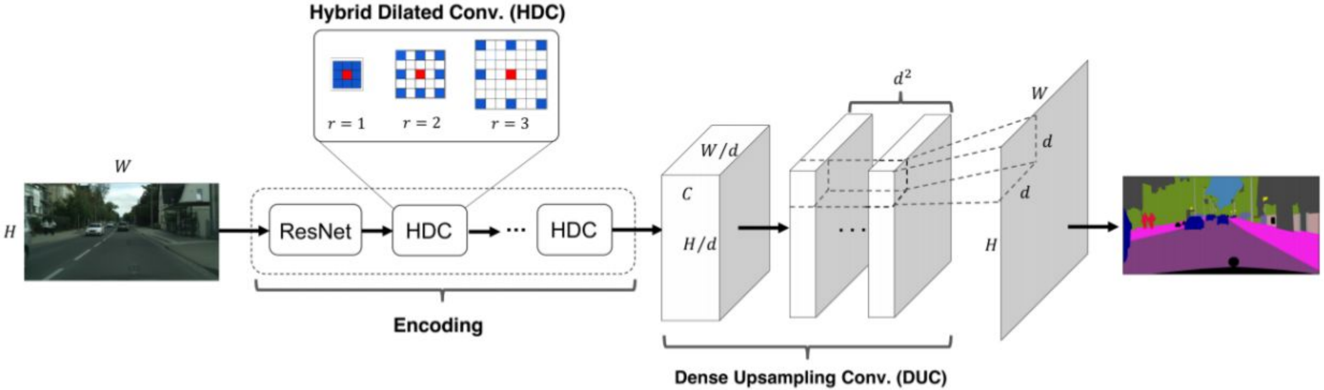

```
ROIs = region_proposal(image)
for ROI in ROIs
    patch = get_patch(image, ROI)
    results = detector(patch)
```

Object Detection / Localization

Single-Shot Methods | Shown: SSD



Semantic Segmentation



Transfer Learning

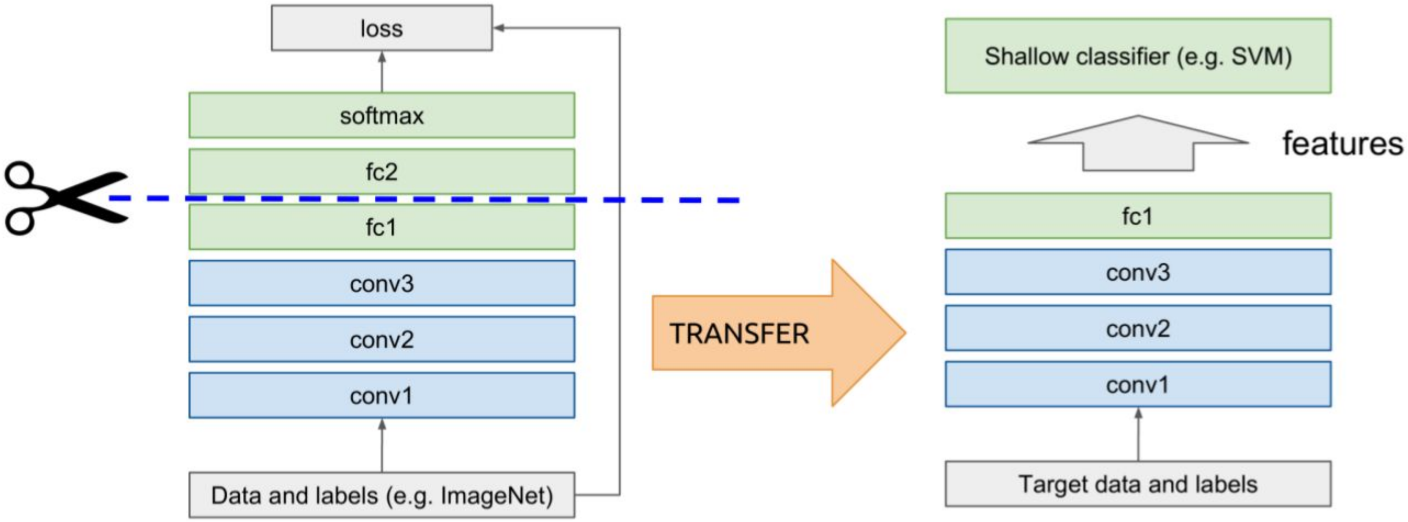
“You need a lot of a data if you want to train/use CNNs”

Transfer Learning

"You need a lot of a data if you want to train/use CNNs"

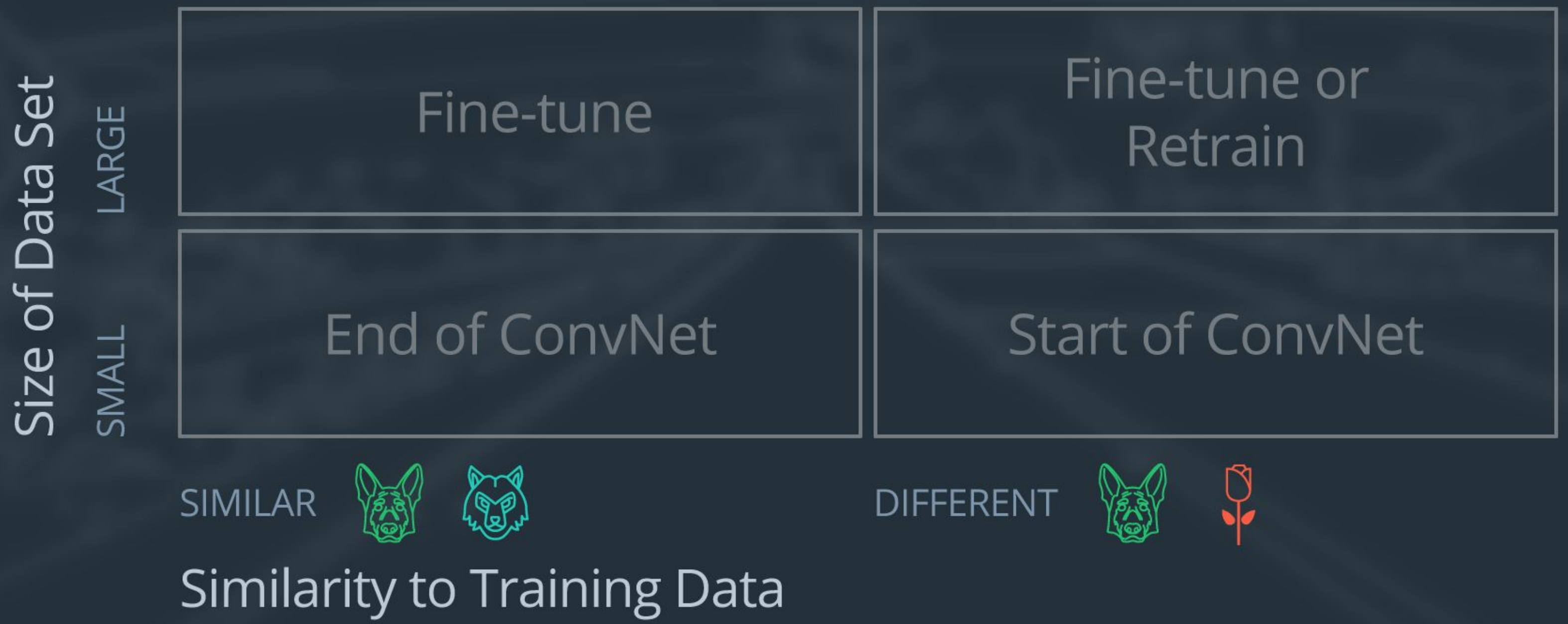
BUSTED

Transfer Learning

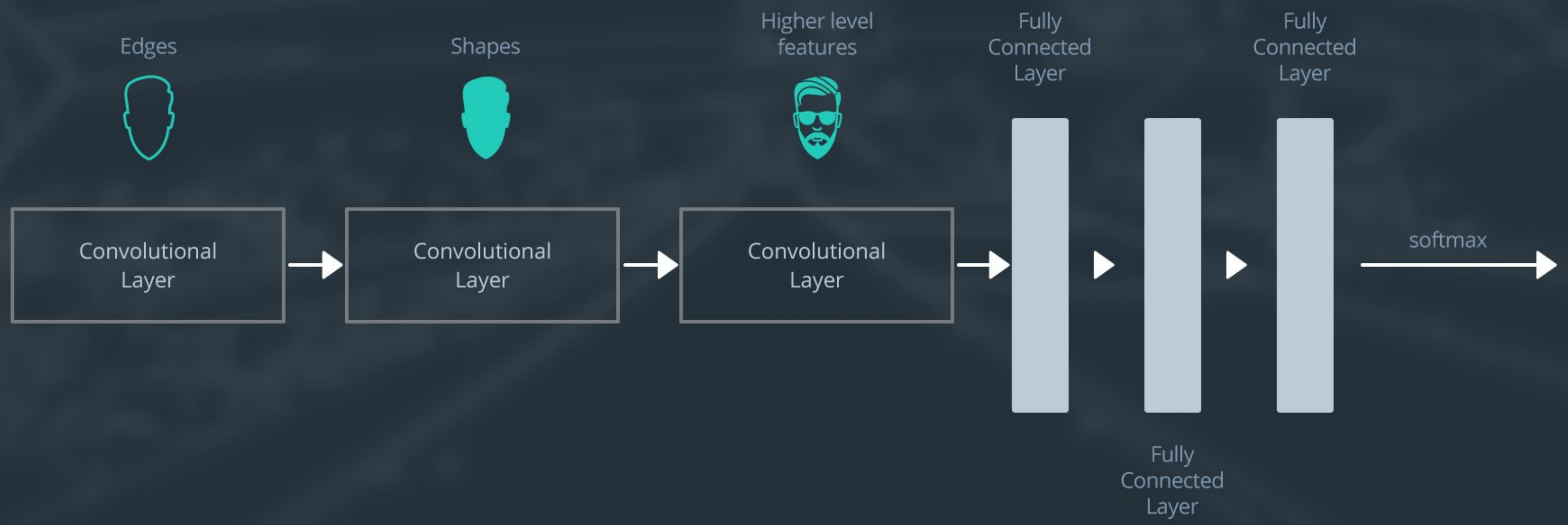


- Fine-tune a pre-trained model
- Effective in many applications: computer vision, audio, speech, natural language processing

Guide for How to Use Transfer Learning



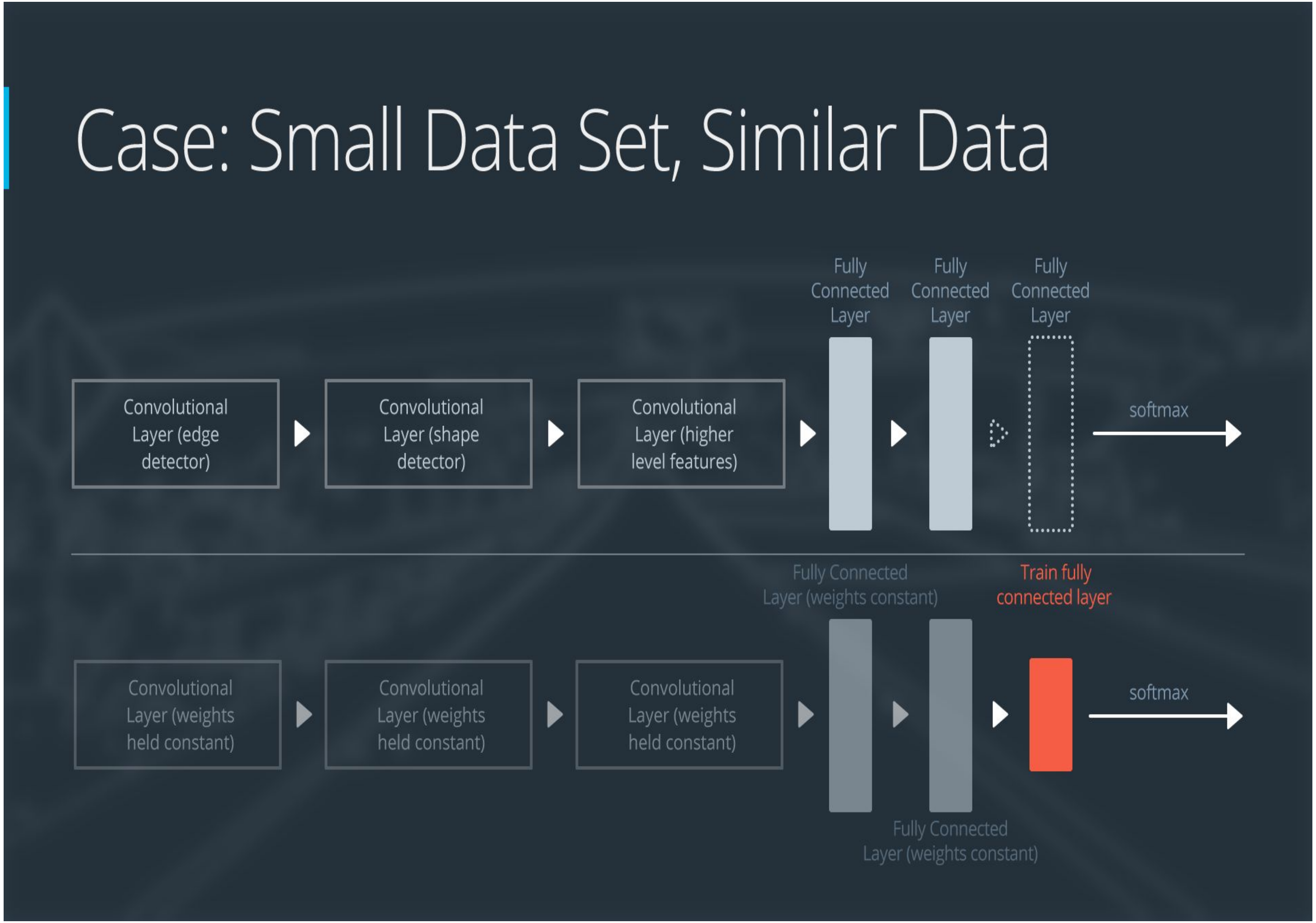
Pre-trained Convolutional Neural Network



Case: Small Data Set, Similar Data

Size of Data Set	SIMILAR	DIFFERENT
LARGE	Fine-tune	Fine-tune or Retrain
SMALL	**End of ConvNet**	Start of ConvNet

Similarity to Training Data



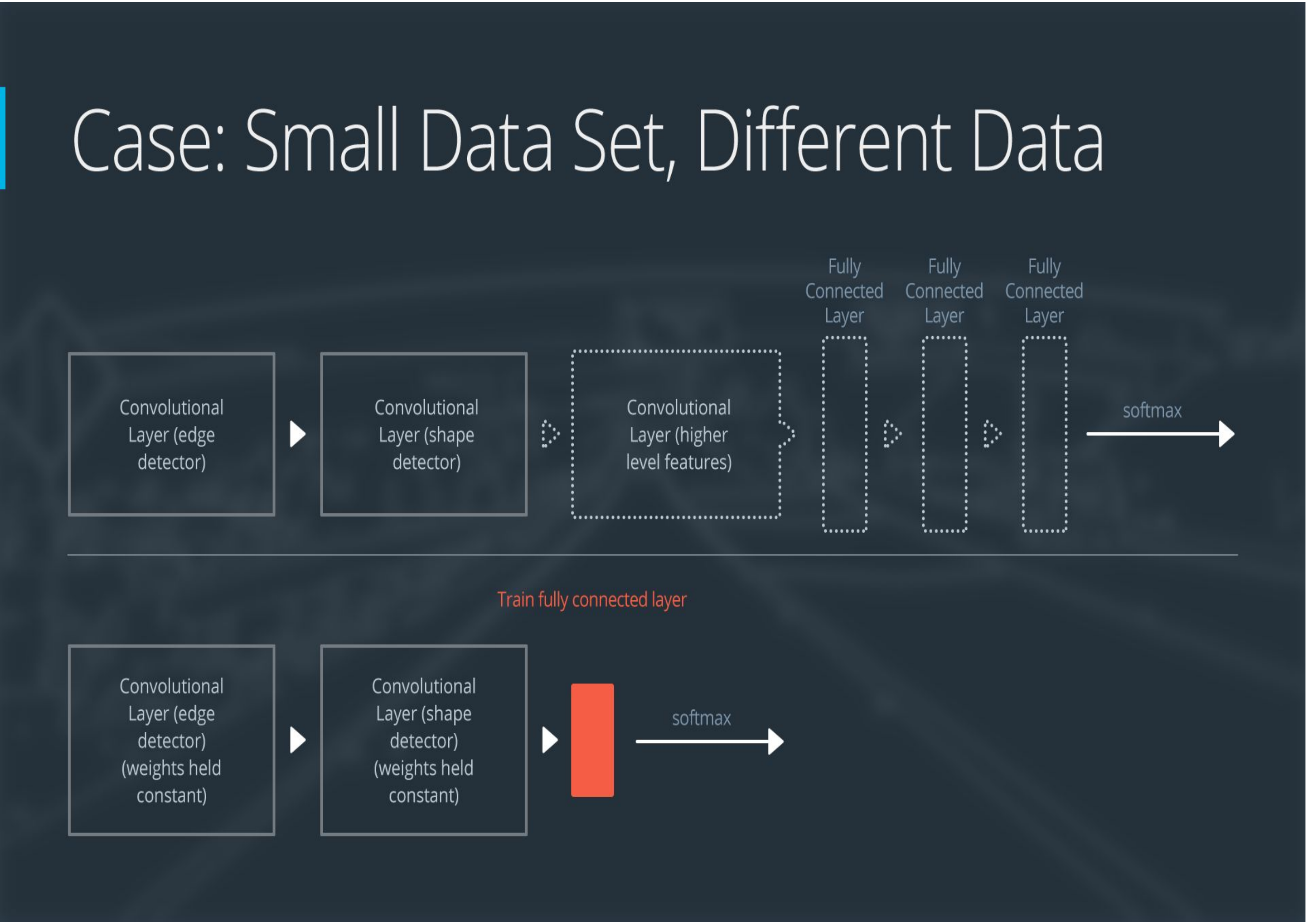
- Quitar la parte final de la red, la última FC. Agregar un FC que matchee con la cantidad de clases e inicializar pesos random. Entrenar solo los pesos de la última FC.

- Congelar los pesos de las capas anteriores para evitar overfitting. Ya que son datasets similares, las imágenes de cada uno van a tener similares high level features. Además, las capas de la red pre-entrenada contiene información relevante y debe mantenerse.

Case: Small Data Set, Different Data

Size of Data Set	SIMILAR	DIFFERENT
LARGE	Fine-tune	Fine-tune or Retrain
SMALL	End of ConvNet	**Start of ConvNet**

Similarity to Training Data

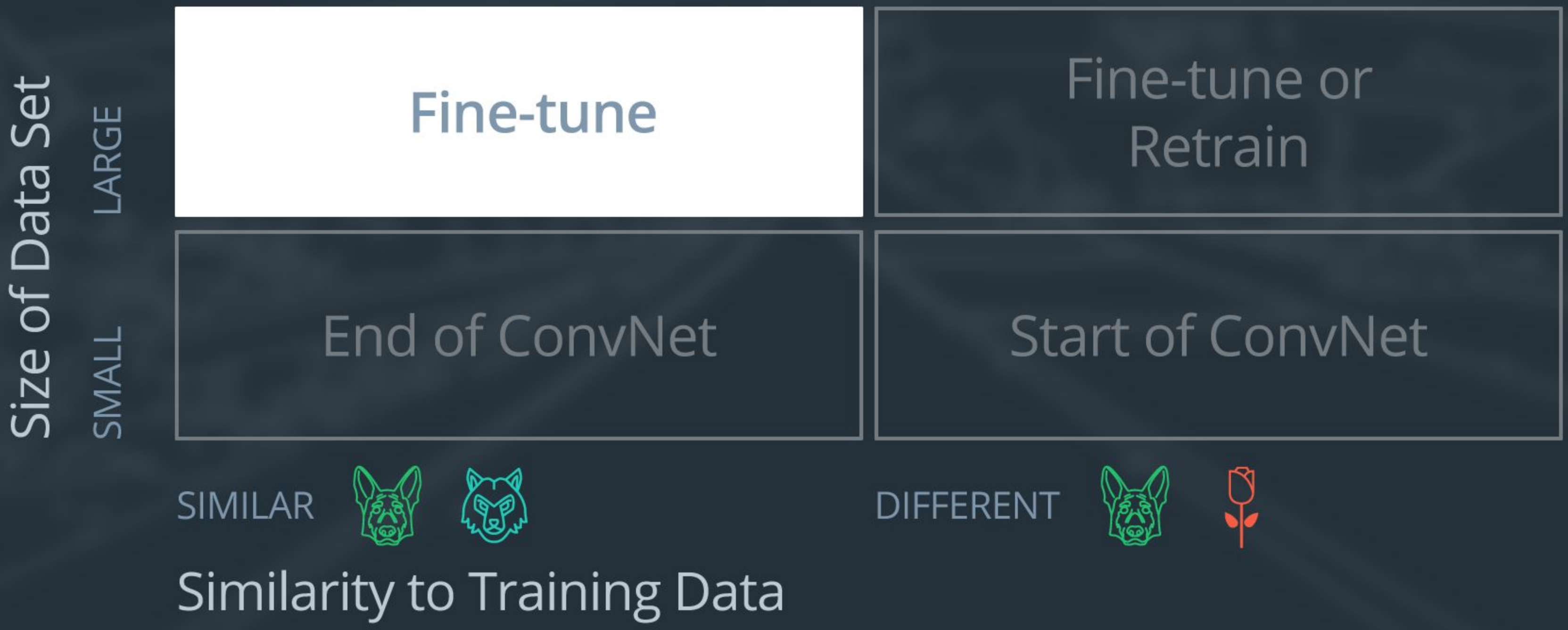


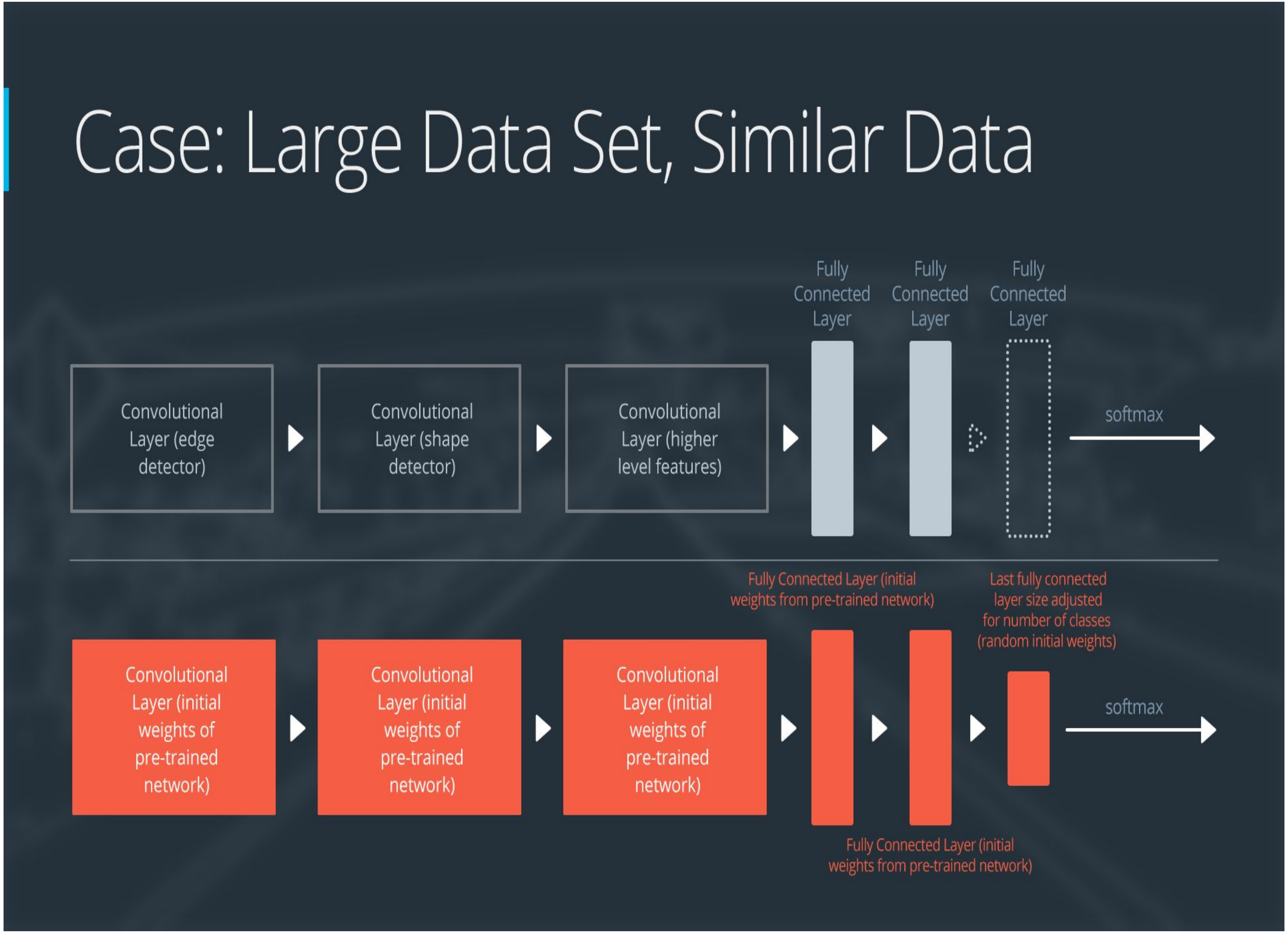
- Quitar la mayoría de capas pre-entrenadas hasta llegar a las características de bajo nivel . Agregar un FC que matchee con la cantidad de clases e inicializar pesos random. Entrenar solo los pesos de la última FC.

- Debido a que el dataset es pequeño existe la posibilidad de overfitting. Para ello se congelan los pesos.

- Al ser diferente data, no comparten características de alto nivel. Por lo que solo usaremos de bajo nivel.

Case: Large Data Set, Similar Data





- Quitar la parte final de la red, la última FC. Agregar un FC que matchee con la cantidad de clases e inicializar pesos random. Entrenar solo los pesos de la última FC.

- Inicializa los pesos de las capas de red pre-entrenada con los pesos pre-entrenados sin congelarlos y re-entrenar el modelo.

- Debido a que los datasets son de clases similares y comparten los mismo high level features, se usa toda la red.

Case: Large Data Set, Different Data

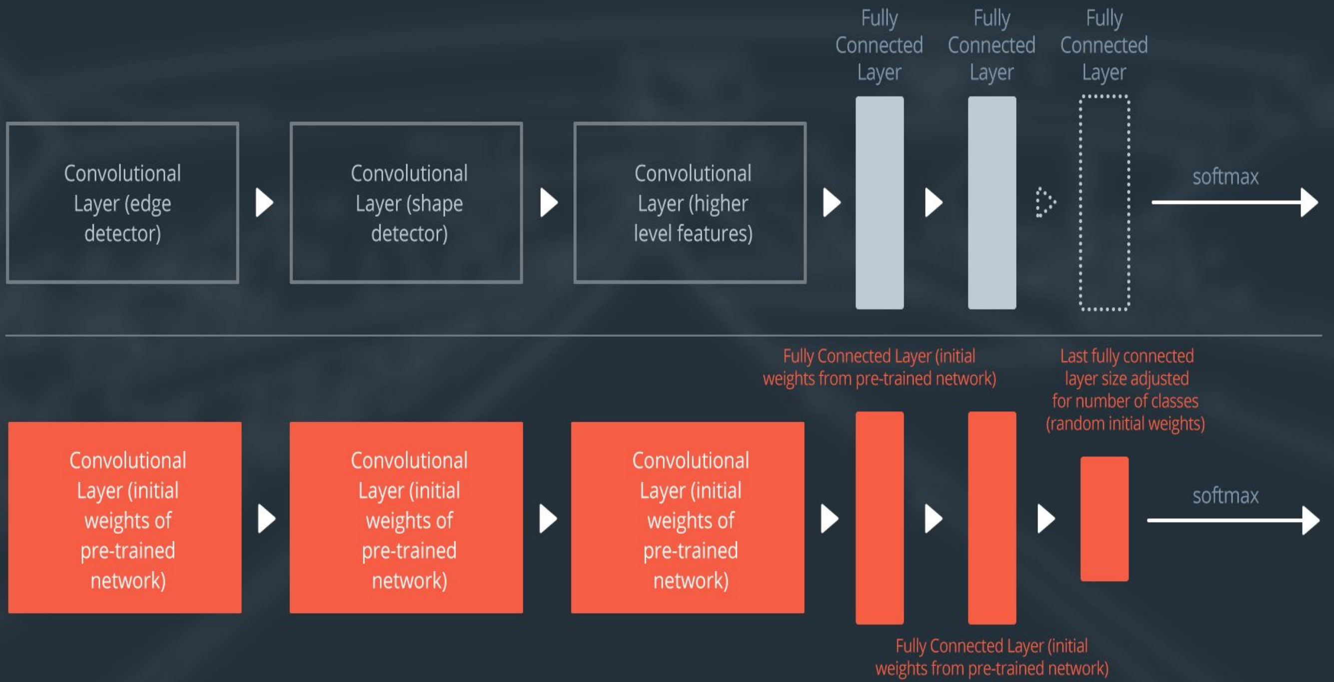
Size of Data Set			
	LARGE	Fine-tune	**Fine-tune or Retrain**
	SMALL	End of ConvNet	Start of ConvNet

SIMILAR 

DIFFERENT 

Similarity to Training Data

Case: Large Data Set, Different Data



- Quitar la parte final de la red, la última FC. Agregar un FC que matchee con la cantidad de clases e inicializar pesos random. Entrenar solo los pesos de la última FC.

- Inicializa los pesos de las capas de red pre-entrenada con pesos pre-entrenados como el caso anterior. En caso no fuera exitoso, iniciar los pesos en random y re-entrenar el modelo.

Notebook

https://github.com/leo2105/Keras_Introduction/blob/master/5_CustomClassifier.ipynb

Issues:

- Usar python 3.6, pip install python=3.6
- Instalar matplotlib, pip install matplotlib
- Crear file data2

Referencias:

- http://cs231n.stanford.edu/
- https://deeplearning.mit.edu/
- https://www.udacity.com/course/deep-learning--ud730

Gracias!