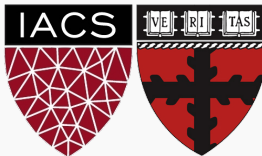


# Advanced Section #3: CNNs and Object Detection

AC 209B: Data Science

Javier Zazo

Pavlos Protopapas



# Lecture Outline

---

Convnets review

Classic Networks

Residual networks

Other combination blocks

Object recognition systems

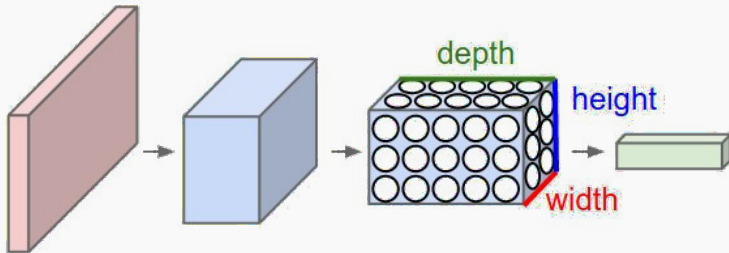
Face recognition systems

## Convnets review

---

# Motivation for convnets

- ▶ Less parameters (weights) than a FC network.
- ▶ Invariant to object translation.
- ▶ Can tolerate some distortion in the images.
- ▶ Capable of generalizing and learning features.
- ▶ Require grid input.

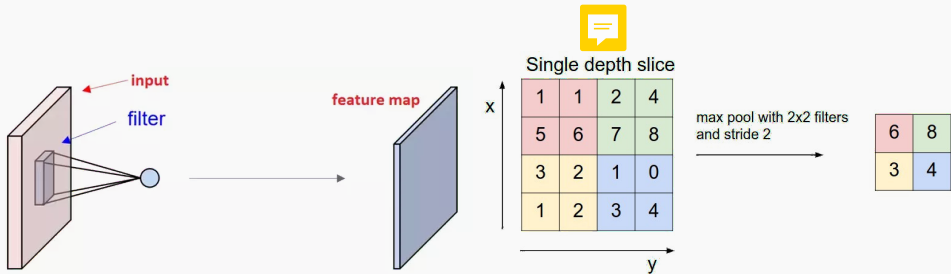


# CNN layers

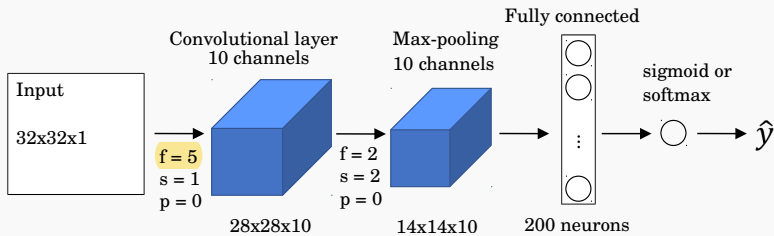
- Convolutional layer: formed by **filters**, **feature maps**, and **activation functions**.
  - Convolution can be *full*, *same* or *valid*.

$$n_{\text{output}} = \left\lfloor \frac{n_{\text{input}} - f + 2p}{s} + 1 \right\rfloor$$

- Pooling layers: reduces overfitting.
- Fully connected layers: mix spacial and channel features together.



# Introductory convolutional network example



► Training parameters:

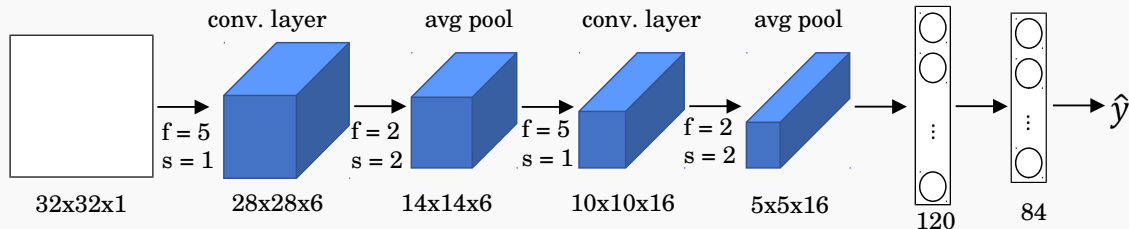
- 250 weights on the conv. filter + 10 bias terms.
- 0 weights on the max-pool.
- $13 \times 13 \times 10 = 1,690$  output elements after max-pool.
- $1,690 \times 200 = 338,000$  weights + 200 bias in the FC layer.
- Total: 338,460 parameters to be trained.

# Classic Networks

---

# LeNet-5

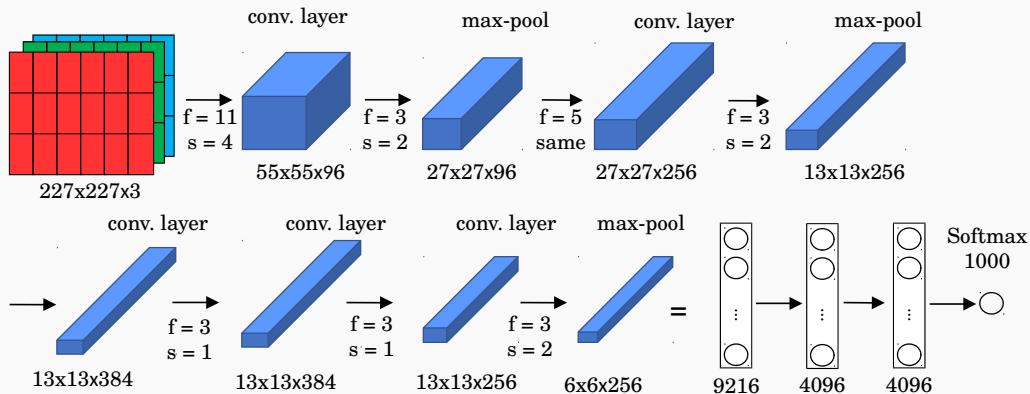
- ▶ Formulation is a bit outdated considering current practices.
- ▶ Uses **convolutional networks** followed by **pooling layers** and finishes with **fully connected layers**.
- ▶ Starts with high dimensional features and reduces their size while increasing the number of channels.
- ▶ Around 60k parameters.





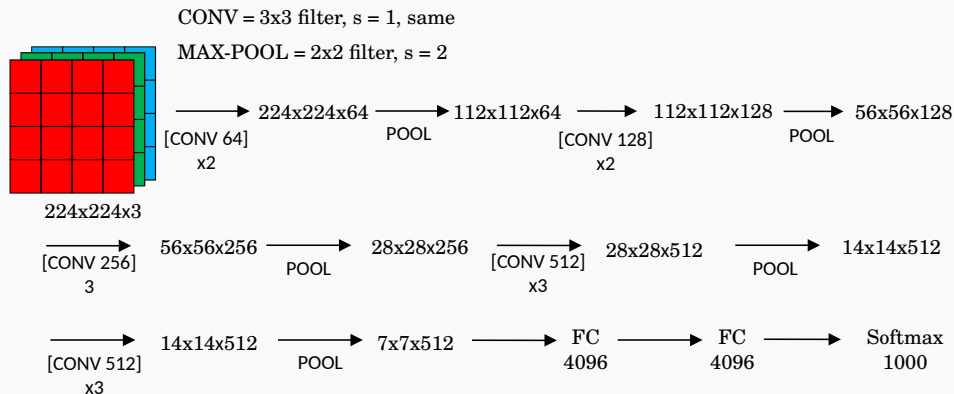
# AlexNet

- ▶ 1.2 million high-resolution ( $227 \times 227 \times 3$ ) images in the ImageNet 2010 contest;
- ▶ 1000 different classes; NN with 60 million parameters to optimize ( $\sim 255$  MB);
- ▶ Uses ReLu activation functions; GPUs for training; 12 layers.



# VGG-16 and VGG-19

- ▶ ImageNet Challenge 2014; 16 or 19 layers; 138 million parameters (522 MB).
- ▶ Convolutional layers use ‘same’ padding and stride  $s = 1$ .
- ▶ Max-pooling layers use a filter size  $f = 2$  and stride  $s = 2$ .

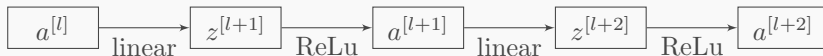


## Residual networks

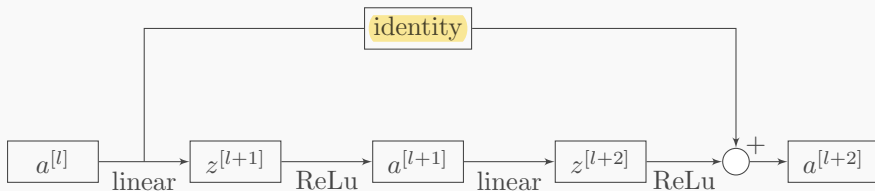
---

# Residual block

- ▶ Residual nets appeared in 2016 to train very deep NN (100 or more layers).
- ▶ Their architecture uses ‘residual blocks’.
- ▶ Plain network structure:



- ▶ **Residual network block:**



---

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun, “Deep residual learning for image recognition,” in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

# Equations of the residual block

## ► Plain network:

$$\begin{aligned}a^{[l]} &= g(z^{[l]}) \\z^{[l+1]} &= W^{[l+1]}a^{[l]} + b^{[l+1]} \\a^{[l+1]} &= g(z^{[l+1]}) \\z^{[l+2]} &= W^{[l+2]}a^{[l+1]} + b^{[l+2]} \\a^{[l+2]} &= g(z^{[l+2]})\end{aligned}$$

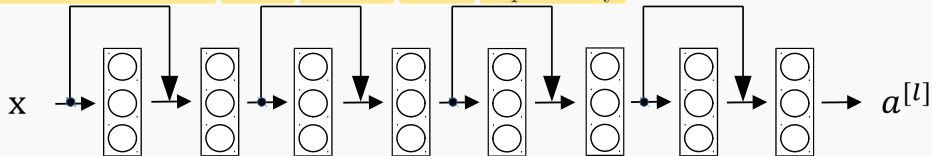
## ► Residual block:

$$\begin{aligned}a^{[l]} &= g(z^{[l]}) \\z^{[l+1]} &= W^{[l+1]}a^{[l]} + b^{[l+1]} \\a^{[l+1]} &= g(z^{[l+1]}) \\z^{[l+2]} &= W^{[l+2]}a^{[l+1]} + b^{[l+2]} \\a^{[l+2]} &= g(z^{[l+2]} + a^{[l]})\end{aligned}$$

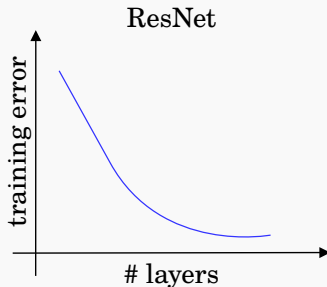
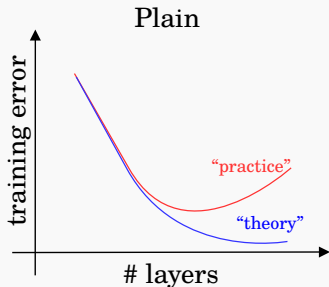
- With this extra connection gradients can travel backwards more easily.
- The residual block can very easily learn the identity function by setting  $W^{[l+2]} = 0$  and  $b^{[l+2]} = 0$ .
- In such case,  $a^{[l+2]} = g(a^{[l]}) = a^{[l]}$  for ReLu units.
  - It becomes a flexible block that can expand the capacity of the network, or simply transform into a identity function that would not affect training.

# Residual network

- ▶ A residual network stacks residual blocks sequentially.



- ▶ The idea is to allow the network to become deeper without increasing the training complexity.

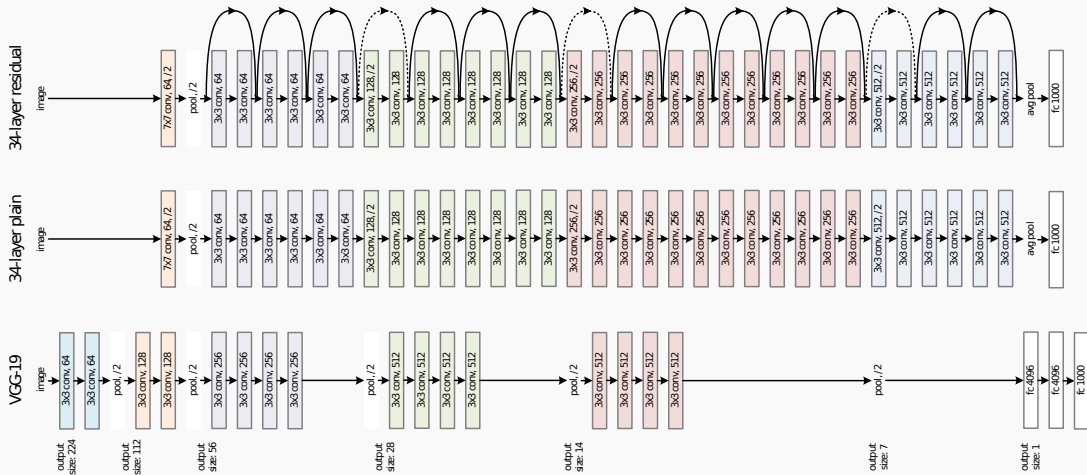


# Residual network

- ▶ Residual networks implement blocks with convolutional layers that use ‘same’ padding option (even when max-pooling).
  - This allows the block to learn the identity function.
- ▶ The designer may want to reduce the size of features and use ‘valid’ padding.
  - In such case, the shortcut path can implement a new set of convolutional layers that reduces the size appropriately.

| Number of Layers | Number of Parameters |
|------------------|----------------------|
| ResNet 18        | 11.174M              |
| ResNet 34        | 21.282M              |
| ResNet 50        | 23.521M              |
| ResNet 101       | 42.513M              |
| ResNet 152       | 58.157M              |

# Residual network 34 layer example





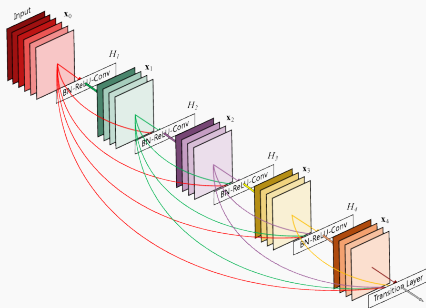
## Classification error values on Imagenet

- ▶ Alexnet (2012) achieved a top-5 error of 15.3% (second place was 26.2%).
- ▶ ZFNet (2013) achieved a top-5 error of 14.8% (visualization of features).

| method                     | top-1 err. | top-5 err.        |
|----------------------------|------------|-------------------|
| VGG [40] (ILSVRC'14)       | -          | 8.43 <sup>†</sup> |
| GoogLeNet [43] (ILSVRC'14) | -          | 7.89              |
| VGG [40] (v5)              | 24.4       | 7.1               |
| PReLU-net [12]             | 21.59      | 5.71              |
| BN-inception [16]          | 21.99      | 5.81              |
| ResNet-34 B                | 21.84      | 5.71              |
| ResNet-34 C                | 21.53      | 5.60              |
| ResNet-50                  | 20.74      | 5.25              |
| ResNet-101                 | 19.87      | 4.60              |
| ResNet-152                 | 19.38      | 4.49              |

# Dense Networks

- **Goal:** allow maximum information (and gradient) flow  $\longrightarrow$  connect every layer directly with each other.
- DenseNets exploit the potential of the network through feature reuse  $\longrightarrow$  no need to learn redundant feature maps.
- DenseNets layers are very narrow (e.g. 12 filters), and they just add a small set of new feature-maps.



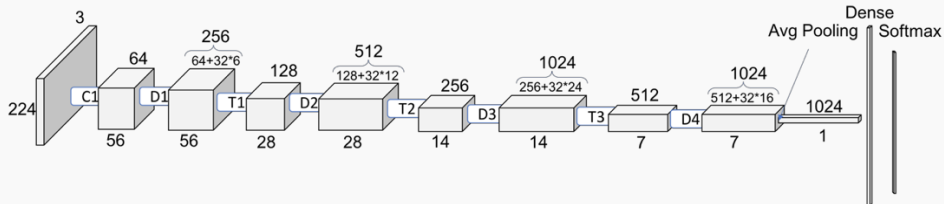
## Dense Networks II

- DenseNets do not sum the output feature maps of the layer with the incoming feature maps but concatenate them:

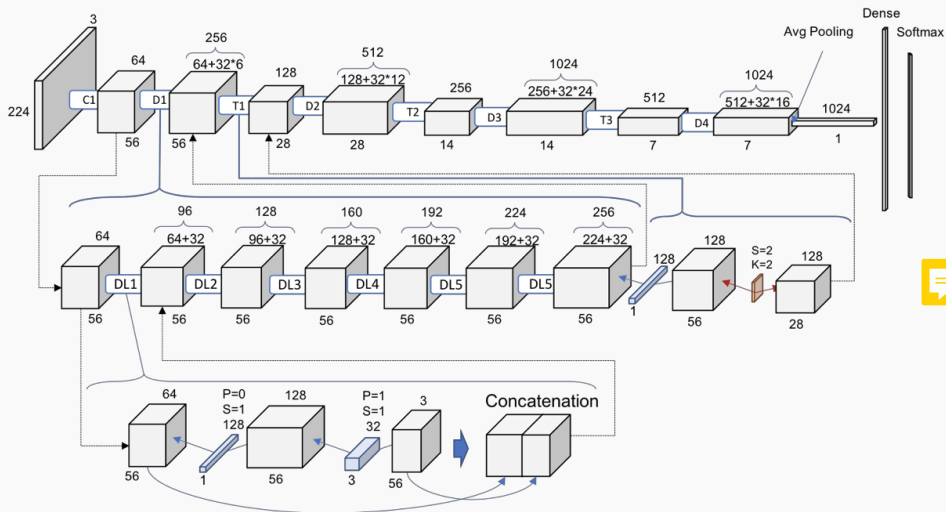
$$a^{[l]} = g([a^{[0]}, a^{[1]}, \dots, a^{[l-1]}])$$

- D dimensions of the feature maps remains constant within a block, but the number of filters changes between them  $\rightarrow$  **growth rate**:

$$k^{[l]} = k^{[0]} + k \cdot (l - 1)$$



# Dense Networks III: Full architecture

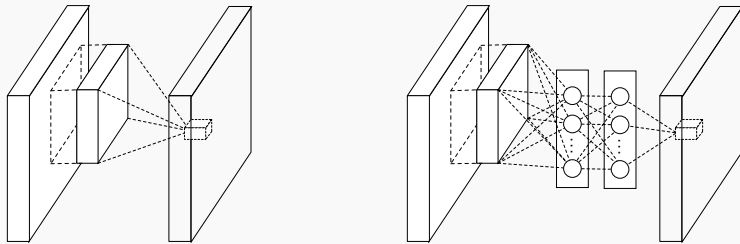


## Other combination blocks

---

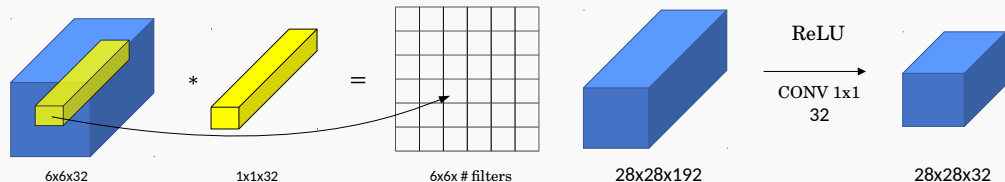
# Network in network

- ▶ Influential concept in the deep learning literature [Lin2013].
- ▶ Authors goal was to generate a deeper network without simply stacking more layers.
- ▶ They replace few filters with a smaller perceptron layers:
  - It is compatible with the backpropagation logic of neural nets.
  - It can itself be a deep model leading to rich separation between latent features.
- ▶ There is a ReLu operation after every neuron:
  - A richer nonlinear function approximator can serve as a better feature extractor.



# 1x1 Convolution

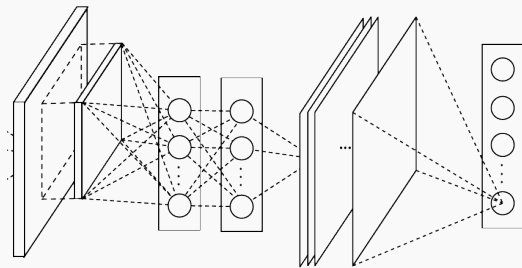
- ▶ A particular case from the previous concept are **1x1 convolutions**.



- ▶ If the input had two dimensions, the  $1 \times 1$  convolution would correspond to a scalar multiplication.
- ▶ With a **greater number of channels** (say, 32), the convolutional filter will have  **$1 \times 1 \times 32$  elements** (more than a simple scaling) + **non-linear activation**.
- ▶ 1x1 convolution leads to **dimension reductionality**  $\rightarrow$  **feature pooling technique**.
  - Reduces the overfitting capacity of the network.
- ▶ FC layers can be regarded as 1x1 convolutions if they go after a FC layer.

# Global Average Pooling

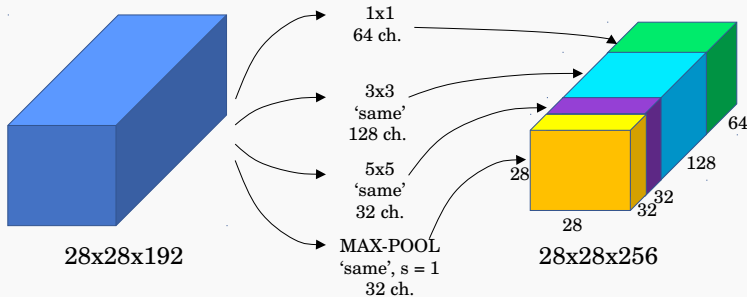
- ▶ Another idea from [Lin2013] is a technique to simplify the last layers of CNNs.
- ▶ In traditional CNNs, feature maps of the last convolution layer are flattened and passed on to one or more fully FC, which are then passed on to softmax.
  - An estimate says that the last FC layers contain 90% of parameters of the NN.
- ▶ Global Average Pooling uses a FC layer with as many outputs as the number of classes being predicted.
- ▶ Then, each map is averaged given rise to the raw scores of the classes and fed to softmax.
  - No new parameters to train (unlike the FC layers), leading to less overfitting.
  - Robust to spatial translations of the input.





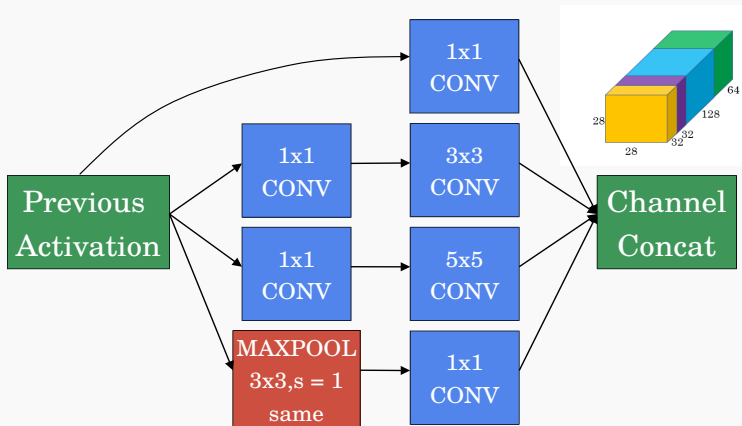
# Inception module

- ▶ The motivation behind inception networks is to use more than a single type of convolutional layer at each layer.
- ▶ Use  $1 \times 1$ ,  $3 \times 3$ ,  $5 \times 5$  convolutional layers, and max-pooling layers in parallel.
- ▶ All modules use *same* convolution.
- ▶ Naïve implementation:



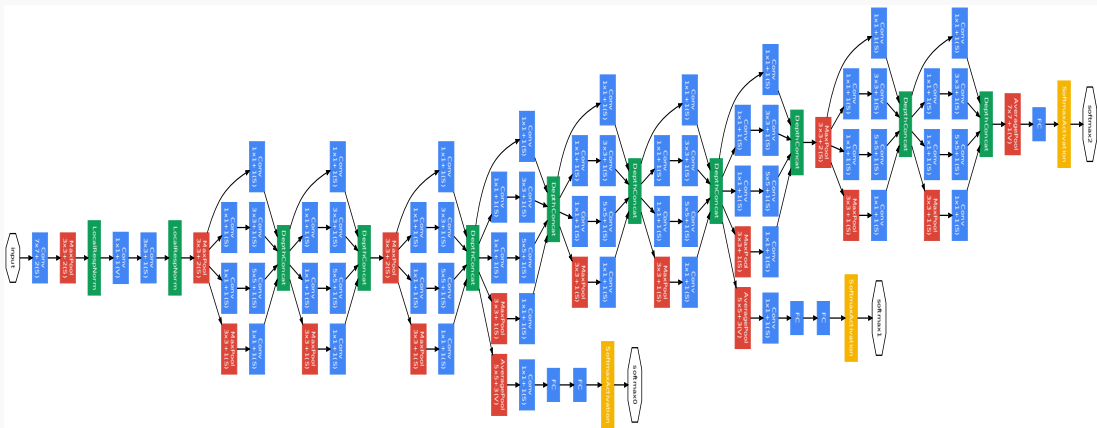
# Inception module with dimension reductions

- Use  $1 \times 1$  convolutions that reduce the size of the channel dimension.
  - The number of channels can vary from the input to the output.



# GoogLeNet network

- ▶ The inception network is formed by concatenating other inception modules.
- ▶ It includes several softmax output units to enforce regularization.



# Summary of networks

- We are now reaching top-5 error rates lower than human manual classification.

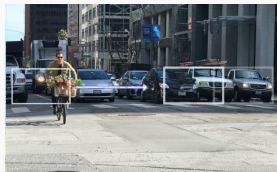
| Year | CNN           | Developed by                                     | Place | Top-5 error rate | No. of parameters |
|------|---------------|--|-------|------------------|-------------------|
| 1998 | LeNet(8)      | Yann LeCun et al                                 |       |                  | 60 thousand       |
| 2012 | AlexNet(7)    | Alex Krizhevsky, Geoffrey Hinton, Ilya Sutskever | 1st   | 15.3%            | 60 million        |
| 2013 | ZFNet()       | Matthew Zeiler and Rob Fergus                    | 1st   | 14.8%            |                   |
| 2014 | GoogLeNet(19) | Google   | 1st   | 6.67%            | 4 million         |
| 2014 | VGG Net(16)   | Simonyan, Zisserman                              | 2nd   | 7.3%             | 138 million       |
| 2015 | ResNet(152)   | Kaiming He                                       | 1st   | 3.6%             |                   |

## Object recognition systems

---

# Sliding-window detectors

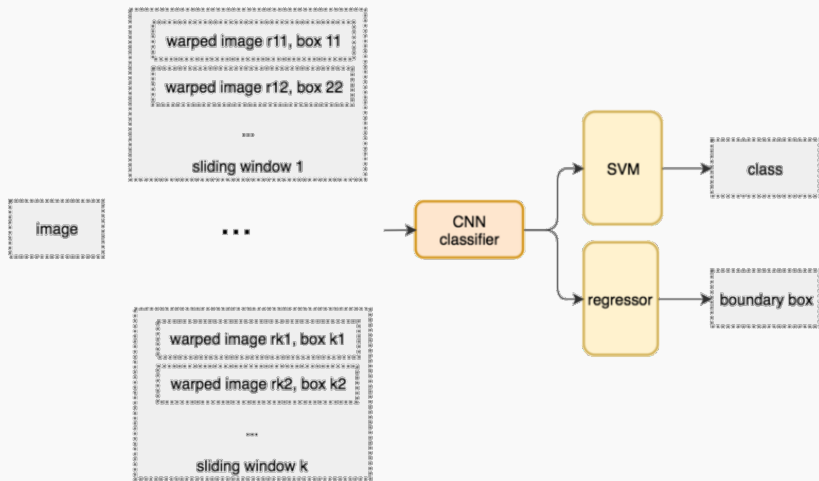
- ▶ Brute force approach → several window sizes moved throughout the image.
- ▶ Patches are cut and warped → passed through a classification CNN.



- ▶ Pseudo-code:

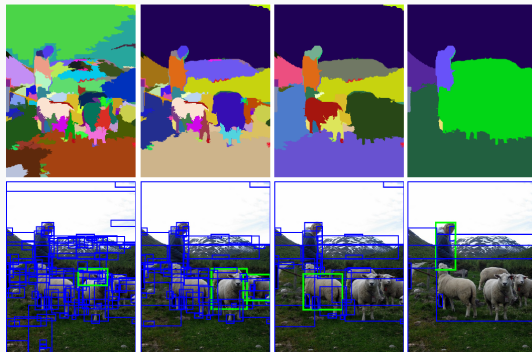
```
for window in windows
    patch = get_patch(image, window)
    results = detector(patch)
```

# Sliding window architecture



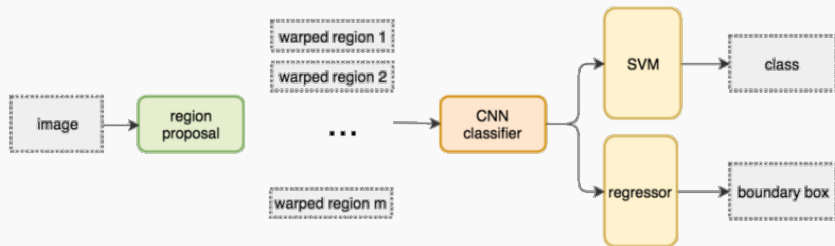
# Selective Search

- ▶ **Goal:** Reduce number of proposed windows → Regions of interest (ROI).
- ▶ Start with single pixel as individual groups → pair groups w/ similarity.
  - Capture all scales: use hierarchical algorithm.
  - Diversification: multiple strategies that consider all use cases.
  - Fast to Compute: should not become a bottleneck?





- Use ROI proposal and feed a CNN.

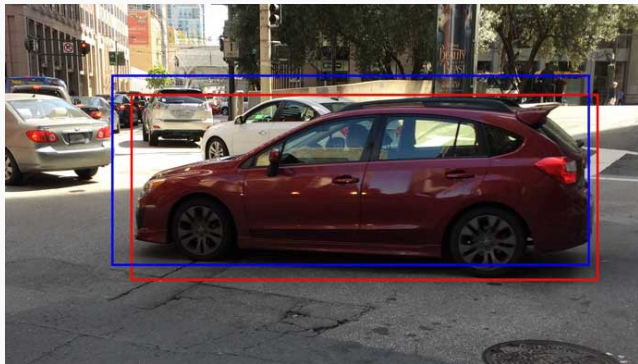


- Pseudo-code:

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

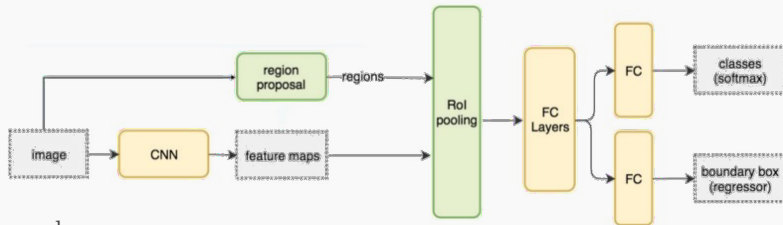
# Boundary box regressor

- ▶ ROI computation is expensive.
- ▶ In order to reduce computation  $\rightarrow$  simplify the ROI proposal.
- ▶ Refine the anchors  $\rightarrow$  FC layer and regression loss.



# Fast R-CNN

- ▶ R-CNN is slow in training & inference → repeat feature extractions 2,000 times.
- ▶ Use a feature extractor (a CNN) to extract features for the whole image first.
- ▶ Warp the patches to a fixed size using ROI pooling and feed them to FC layers.



- ▶ Pseudo-code:

```
feature_maps = process(image)
ROIs = region_proposal(image)
for ROI in ROIs
    patch = roi_pooling(feature_maps, ROI)
    results = detector2(patch)
```

# ROI Pooling

- Perform max-pooling operations on feature maps for regions of different sizes.

|     |     |     |     |     |     |     |     |
|-----|-----|-----|-----|-----|-----|-----|-----|
| 0.1 | 0.3 | 0.2 | 0.3 | 0.2 | 0.6 | 0.8 | 0.9 |
| 0.4 | 0.5 | 0.1 | 0.4 | 0.7 | 0.1 | 0.4 | 0.3 |
| 0.2 | 0.1 | 0.3 | 0.8 | 0.6 | 0.2 | 0.1 | 0.1 |
| 0.4 | 0.6 | 0.2 | 0.1 | 0.3 | 0.6 | 0.1 | 0.2 |
| 0.1 | 0.8 | 0.3 | 0.3 | 0.5 | 0.3 | 0.3 | 0.3 |
| 0.2 | 0.9 | 0.4 | 0.5 | 0.1 | 0.1 | 0.1 | 0.2 |
| 0.3 | 0.1 | 0.8 | 0.6 | 0.3 | 0.3 | 0.6 | 0.5 |
| 0.5 | 0.5 | 0.2 | 0.1 | 0.1 | 0.2 | 0.1 | 0.2 |

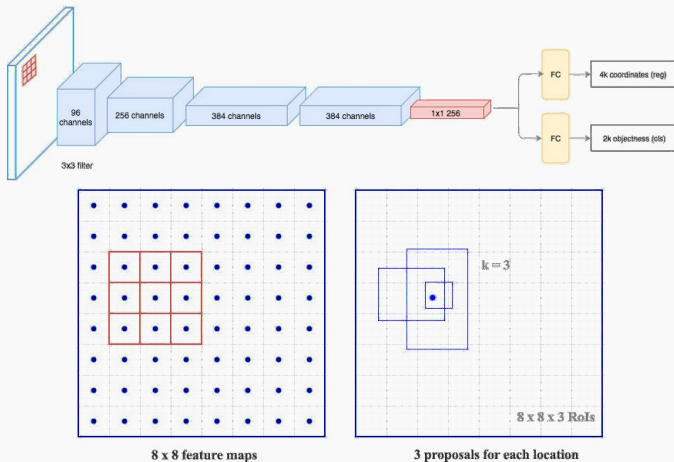
|     |     |     |     |     |     |     |     |
|-----|-----|-----|-----|-----|-----|-----|-----|
| 0.1 | 0.3 | 0.2 | 0.3 | 0.2 | 0.6 | 0.8 | 0.9 |
| 0.4 | 0.5 | 0.1 | 0.4 | 0.7 | 0.1 | 0.4 | 0.3 |
| 0.2 | 0.1 | 0.3 | 0.8 | 0.6 | 0.2 | 0.1 | 0.1 |
| 0.4 | 0.6 | 0.2 | 0.1 | 0.3 | 0.6 | 0.1 | 0.2 |
| 0.1 | 0.8 | 0.3 | 0.3 | 0.5 | 0.3 | 0.3 | 0.3 |
| 0.2 | 0.9 | 0.4 | 0.5 | 0.1 | 0.1 | 0.1 | 0.2 |
| 0.3 | 0.1 | 0.8 | 0.6 | 0.3 | 0.3 | 0.6 | 0.5 |
| 0.5 | 0.5 | 0.2 | 0.1 | 0.1 | 0.2 | 0.1 | 0.2 |

|     |     |     |     |     |     |     |     |
|-----|-----|-----|-----|-----|-----|-----|-----|
| 0.1 | 0.3 | 0.2 | 0.3 | 0.2 | 0.6 | 0.8 | 0.9 |
| 0.4 | 0.5 | 0.1 | 0.4 | 0.7 | 0.1 | 0.4 | 0.3 |
| 0.2 | 0.1 | 0.3 | 0.8 | 0.6 | 0.2 | 0.1 | 0.1 |
| 0.4 | 0.6 | 0.2 | 0.1 | 0.3 | 0.6 | 0.1 | 0.2 |
| 0.1 | 0.8 | 0.3 | 0.3 | 0.5 | 0.3 | 0.3 | 0.3 |
| 0.2 | 0.9 | 0.4 | 0.5 | 0.1 | 0.1 | 0.1 | 0.2 |
| 0.3 | 0.1 | 0.8 | 0.6 | 0.3 | 0.3 | 0.6 | 0.5 |
| 0.5 | 0.5 | 0.2 | 0.1 | 0.1 | 0.2 | 0.1 | 0.2 |

|     |     |
|-----|-----|
| 0.8 | 0.6 |
| 0.9 | 0.6 |

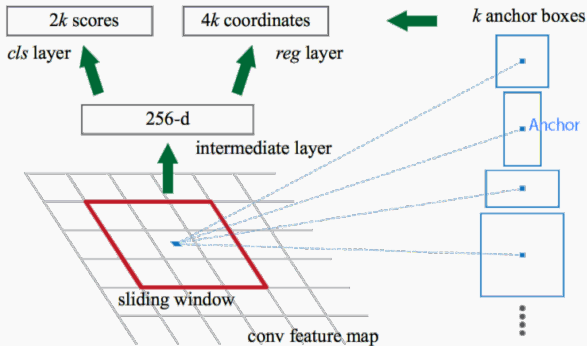
# Faster R-CNN

- Substitute the **region proposal** with a **Region proposal network (RPN)**.
- For each location in the feature maps, RPN makes  $k$  guesses.



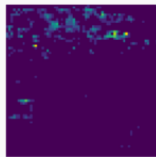
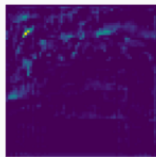
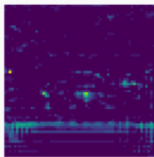
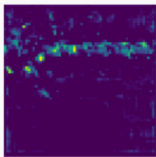
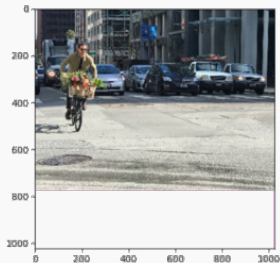
# RPN (Region Proposal Network)

- Faster R-CNN uses far more anchors. It deploys 9 anchor boxes: 3 different scales at 3 different aspect ratio. Using 9 anchors per location, it generates 2 9 objectness scores and 4 9 coordinates per location.



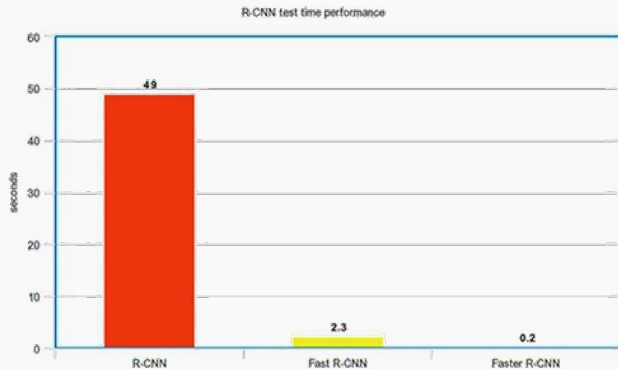
# Visualizing ROI proposals

1. Perform ROI proposals (RPN or distance algorithm)  $\rightarrow$  feed to CNN.
2. Output boundary box (refinements) and objectiveness score.
3. **Perform per class non-maximum suppression**  $\rightarrow$  removes duplicate objects.



# Performance for R-CNN methods

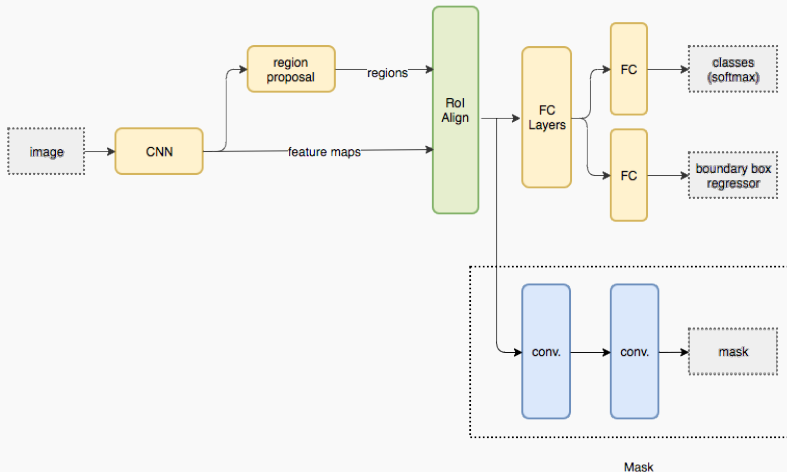
- Faster R-CNN is even much faster.





# Mask R-CNN

- From Faster R-CNN we add another CNN to mask regions.
- The additional CNN mask is only for coloring.



# ROI Align

- Refinement of the ROI pooling.
- Makes every target cell to have the same size.
- It also applies interpolation.

|     |     |     |     |     |     |     |     |
|-----|-----|-----|-----|-----|-----|-----|-----|
| 0.1 | 0.3 | 0.2 | 0.3 | 0.2 | 0.6 | 0.8 | 0.9 |
| 0.4 | 0.5 | 0.1 | 0.4 | 0.7 | 0.1 | 0.4 | 0.3 |
| 0.2 | 0.1 | 0.3 | 0.8 | 0.6 | 0.2 | 0.1 | 0.1 |
| 0.4 | 0.6 | 0.2 | 0.1 | 0.3 | 0.6 | 0.1 | 0.2 |
| 0.1 | 0.8 | 0.3 | 0.3 | 0.5 | 0.3 | 0.3 | 0.3 |
| 0.2 | 0.9 | 0.4 | 0.5 | 0.1 | 0.1 | 0.1 | 0.2 |
| 0.3 | 0.1 | 0.8 | 0.6 | 0.3 | 0.3 | 0.6 | 0.5 |
| 0.5 | 0.5 | 0.2 | 0.1 | 0.1 | 0.2 | 0.1 | 0.2 |

|     |     |     |     |     |     |     |     |
|-----|-----|-----|-----|-----|-----|-----|-----|
| 0.1 | 0.3 | 0.2 | 0.3 | 0.2 | 0.6 | 0.8 | 0.9 |
| 0.4 | 0.5 | 0.1 | 0.4 | 0.7 | 0.1 | 0.4 | 0.3 |
| 0.2 | 0.1 | 0.3 | 0.8 | 0.6 | 0.2 | 0.1 | 0.1 |
| 0.4 | 0.6 | 0.2 | 0.1 | 0.3 | 0.6 | 0.1 | 0.2 |
| 0.1 | 0.8 | 0.3 | 0.3 | 0.5 | 0.3 | 0.3 | 0.3 |
| 0.2 | 0.9 | 0.4 | 0.5 | 0.1 | 0.1 | 0.1 | 0.2 |
| 0.3 | 0.1 | 0.8 | 0.6 | 0.3 | 0.3 | 0.6 | 0.5 |
| 0.5 | 0.5 | 0.2 | 0.1 | 0.1 | 0.2 | 0.1 | 0.2 |

|     |     |
|-----|-----|
| 0.8 | 0.6 |
| 0.9 | 0.6 |

|      |     |
|------|-----|
| 0.88 | 0.6 |
| 0.9  | 0.6 |

|                 | AP          | AP <sub>50</sub> | AP <sub>75</sub> | AP <sup>bb</sup> | AP <sup>bb</sup> <sub>50</sub> | AP <sup>bb</sup> <sub>75</sub> |
|-----------------|-------------|------------------|------------------|------------------|--------------------------------|--------------------------------|
| <i>RoIPool</i>  | 23.6        | 46.5             | 21.6             | 28.2             | 52.7                           | 26.9                           |
| <i>RoIAlign</i> | <b>30.9</b> | <b>51.8</b>      | <b>32.1</b>      | <b>34.0</b>      | <b>55.3</b>                    | <b>36.4</b>                    |
|                 | +7.3        | +5.3             | +10.5            | +5.8             | +2.6                           | +9.5                           |

# Single Shot Detectors

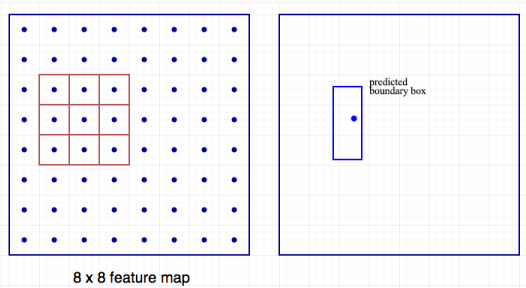
- ▶ **Goal:** Do not generate ROI proposals.

```
feature_maps = process(image)
results = detector(feature_maps)           # No more separate step for ROIs.
```

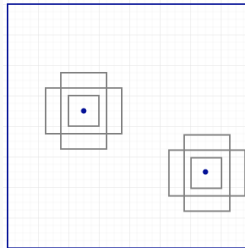
- ▶ Sliding windows/ROIs requires too many shapes to cover most objects.
- ▶ Then use detectors to predict class and boundary box → this is expensive.
- ▶ Single shot detectors predict both boundary box and class at the same time.
- ▶ These networks are trained end to end → they are very fast, and increases accuracy w.r.t. to purpose oriented subnetworks.
- ▶ Single shot detector often trades accuracy with real-time processing speed.

# Single Shot Detectors II

► ROI based:



► Single Shot:

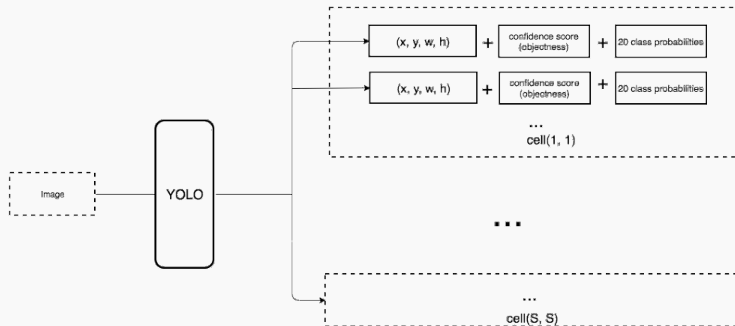


# YOLO Architecture

- ▶ Number of predicted parameters on output feature maps  $8 \times 8$ :

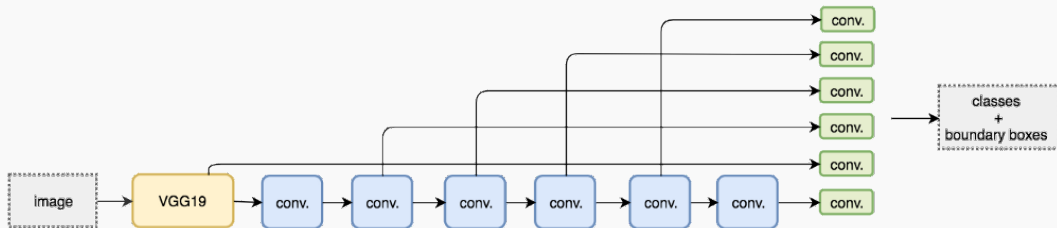
$$8 \times 8 \times D \longrightarrow (S, S, B \cdot 5 + C) = 8 \times 8 \times B \times 5 + 20 \text{ for } C=20.$$

- ▶ YOLO has evolved to YOLOv2, YOLO9000 and YOLOv3, with improvements (such as multiscale, multiple box predictions, location box, word tree search, FPN, etc.).



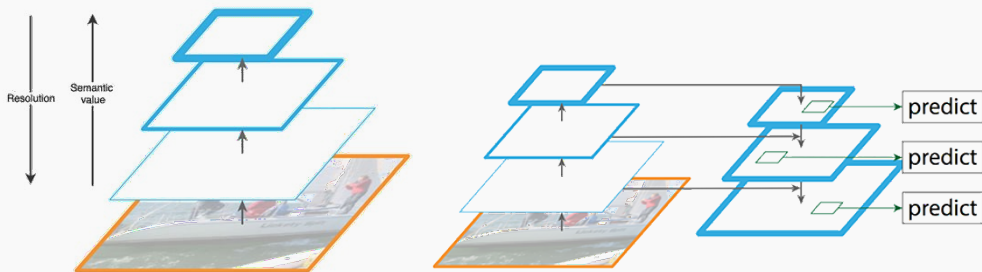
# SSD Architecture

- ▶ Uses a CNN as a feature extractor  $\rightarrow$  same as Faster R-CNN.
- ▶ Then add custom convolution layers to make predictions.
- ▶ Previous model can detect large objects only  $\rightarrow$  make independent object detections from multiple feature maps.



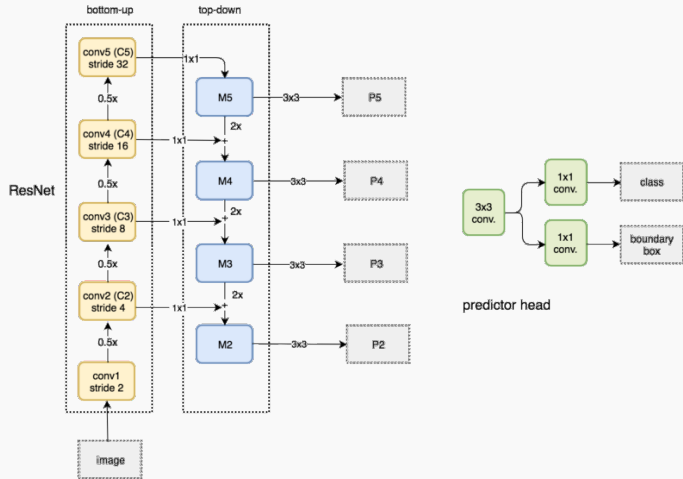
# Feature Pyramid Networks (FPN)

- ▶ Feature extractor to improve accuracy and speed.
- ▶ It helps to generate higher quality features.
- ▶ On higher layers the semantic value increases; vs. spatial resolution.
- ▶ We can mix the information flow.



# Feature Pyramid Networks (FPN) II

- FPNs can work with object detectors → RPNs and classifiers:





## Face recognition systems

---

# Face recognition systems

---

- ▶ Verification

- **Input:** Image from a person to identify and a ID.
- **Objective:** decide whether the input image corresponds to the ID.

- ▶ Recognition

- **Database** of  $K$  people.
- **Input:** Image from a person to identify.
- **Objective:** Identify the person in the database or reject recognition.

- ▶ Recognition is a much harder problem than verification for a specified performance.

# One-shot learning

---

## Verification:

- ▶ We only have a single photo to learn the characteristics of a given person.
- ▶ Then, given a new photo, output if they correspond to the same person.
- ▶ We can construct a similarity function or distance between images:

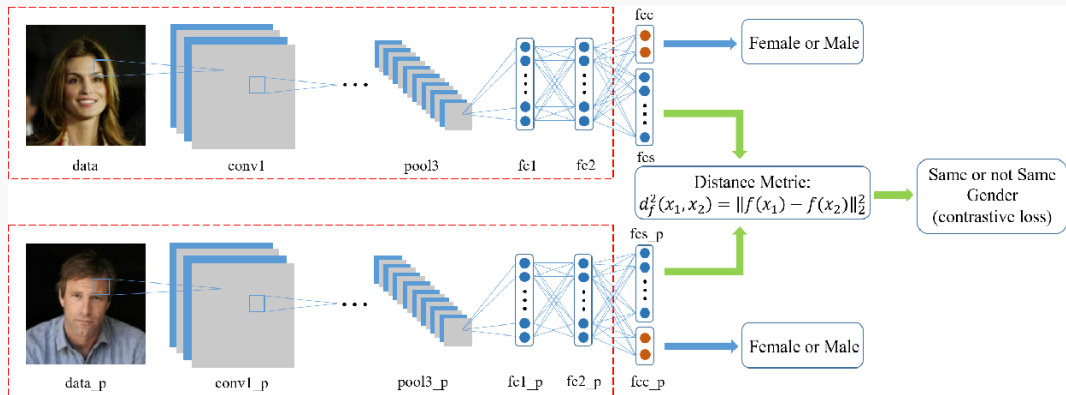
$$d(\text{img1}, \text{img2})$$

- Then, set a threshold to balance accuracy and precision.

# Siamese network

- Build a NN to generate a latent representation of an image.
- Perform two independent calculations on the input.
- Construct a loss function to determine distance between latent features:

$$f(x, y) = \|f(x) - f(y)\|^2 = \|a_x^{[L]} - a_y^{[L]}\|^2$$



# Loss functions

---

- ▶ Loss should be small for the same person and far apart for different people.
- ▶ Use cross-entropy and define:

$$f(x, y) = \sum_i w_i |a_i^{[L](x)} - a_i^{[L](y)}| + b_i$$

- ▶  $\chi^2$  loss:

$$f(x, y) = \sum_i w_i \frac{(a_i^{[L](x)} - a_i^{[L](y)})^2}{(a_i^{[L](x)} + a_i^{[L](y)})}$$

- The representations in DeepFace are normalized between 0 and 1 to reduce the sensitivity to illumination changes.

# Triplet loss

- ▶ Given three images A, P, N:

$$\|f(A) - f(P)\|^2 + \alpha < \|f(A) - f(N)\|^2$$

$$\mathcal{L}(A, P, N) = \min (\|f(A) - f(P)\|^2 - \|f(A) - f(N)\|^2 + \alpha, 0)$$

- ▶ Training:



- ▶ Evaluation:

$$f(x, y) = \|a_x^{[L]} - a_y^{[L]}\|^2 \leq \tau$$

- ▶ Train on 10k pictures of 1k persons.
- ▶ Need to choose triplets that are “hard to train on.”

Thank you!

---

Questions?