CS409 : Neural Networks (Semester II - 2021/22)

Unit 7: Convolutional Neural Networks (CNNs) (2)

Dr. Ruwan Nawarathna
Department of Statistics & Computer Science
Faculty of Science
University of Peradeniya

CNNs for Image Classification in Computer Vision

- Fashion MNIST Example
  - Refer the code discussed in the class

Popular CNNs

AlexNet (Krizhevsky, Sutskever, & Hinton, 2012)

- The AlexNet CNN architecture won the 2012 ImageNet ILSVRC challenge by a large margin. It achieved a 17% top-5 error rate while the second-best achieved only 26%!
  - Turning point of Neural Networks

- It was developed by Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton.

What is ImageNet?

- > 15M high resolution images
- over 22K categories
- labeled by Mechanical Turk workers
- E.g., fungi

ImageNet Large-Scale Visual Recognition Challenge (ILSVRC)

- 2010-2017
  - multiple challenges
    - classification
    - classification and localization
    - segmentation
- 2010 Classification Challenge
  - 1.2 M training images, 1000 categories (general and specific)
  - 200K test images
  - output a list of 5 object categories in descending order of confidence
  - two error rates: top-1 and top-5

ILSVRC - Winners

YEAR	WINNER	TOP 5 ERROR RATE %
2012	ALEXNET	15.3
2013	ZFNET	11.2
2014	INCEPTION V1 (GoogLeNet) VGG NET (Runner up)	6.67 7.3
2015	ResNet	3.57
2016	ResNeXt	4.1
2017	SENet	2.251
2018	PNASNet-5	3.8

AlexNet

- Architecture
  - 5 convolutional layers, split across two GPUs
  - 2 fully connected layers
  - 1000-way softmax output layer
- Trained with SGD for ~ 1 week
  - 650k neurons
  - 60M parameters
  - 630M connections

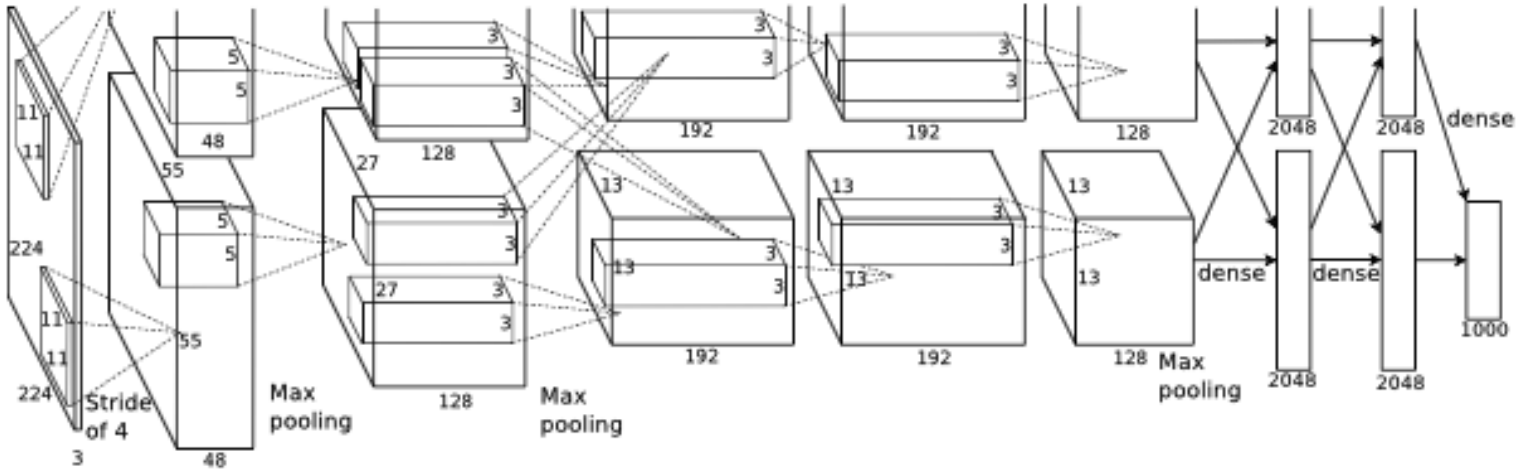
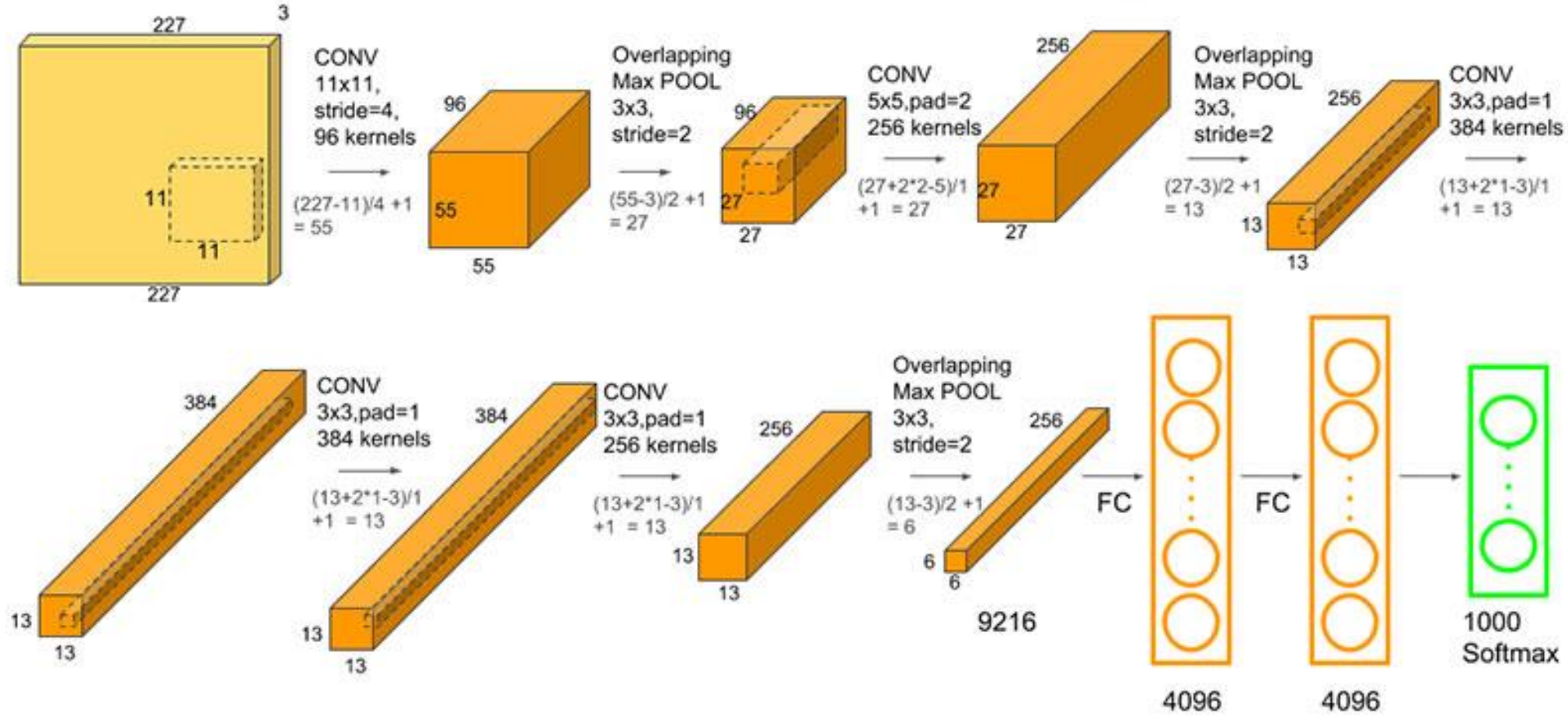


Figure 2: An illustration of the architecture of our CNN, explicitly showing the delineation of responsibilities between the two GPUs. One GPU runs the layer-parts at the top of the figure while the other runs the layer-parts at the bottom. The GPUs communicate only at certain layers. The network's input is 150,528-dimensional, and the number of neurons in the network's remaining layers is given by 253,440–186,624–64,896–64,896–43,264–4096–4096–1000.

The Architecture of AlexNet

Full (simplified) AlexNet architecture:
[227x227x3] INPUT
[55x55x96] CONV1: 96 11x11 filters at stride 4, pad 0
[27x27x96] MAX POOL1: 3x3 filters at stride 2
[27x27x96] NORM1: Normalization layer
[27x27x256] CONV2: 256 5x5 filters at stride 1, pad 2
[13x13x256] MAX POOL2: 3x3 filters at stride 2
[13x13x256] NORM2: Normalization layer
[13x13x384] CONV3: 384 3x3 filters at stride 1, pad 1
[13x13x384] CONV4: 384 3x3 filters at stride 1, pad 1
[13x13x256] CONV5: 256 3x3 filters at stride 1, pad 1
[6x6x256] MAX POOL3: 3x3 filters at stride 2
[4096] FC6: 4096 neurons
[4096] FC7: 4096 neurons
[1000] FC8: 1000 neurons (class scores)

The Architecture of AlexNet



VGGNet

- VGGNet. The runner-up in ILSVRC 2014 was the network from Karen Simonyan and Andrew Zisserman that became known as the VGGNet

Architecture of VGGNet

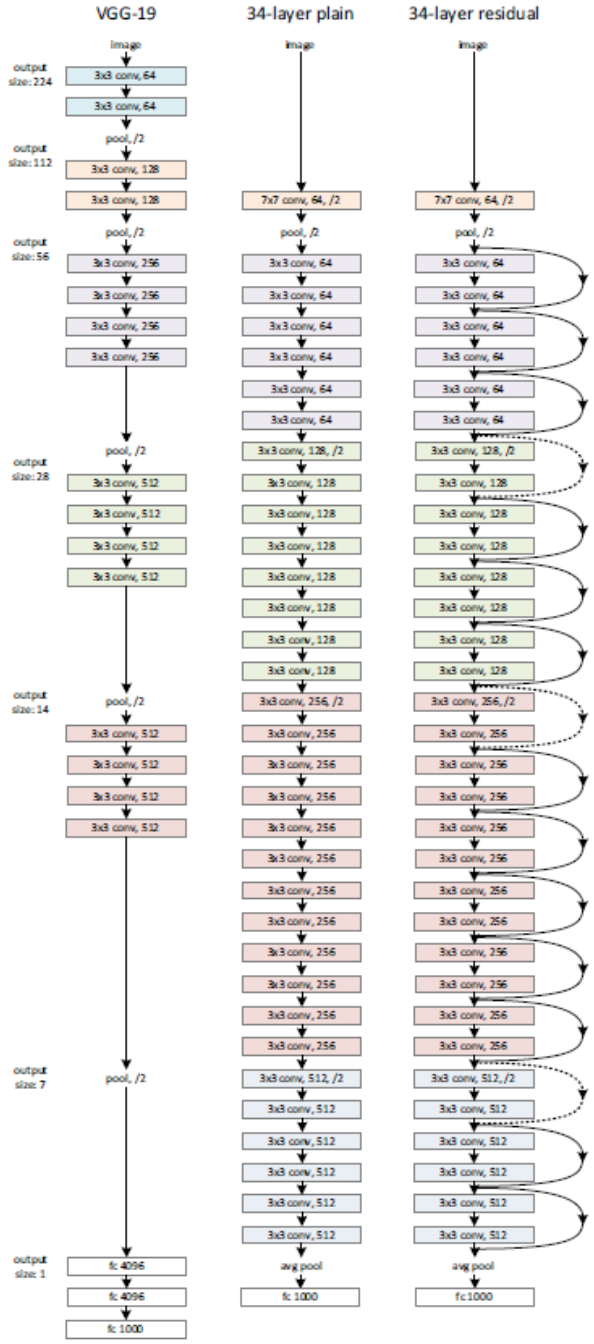
VGG16

VGGNet 16

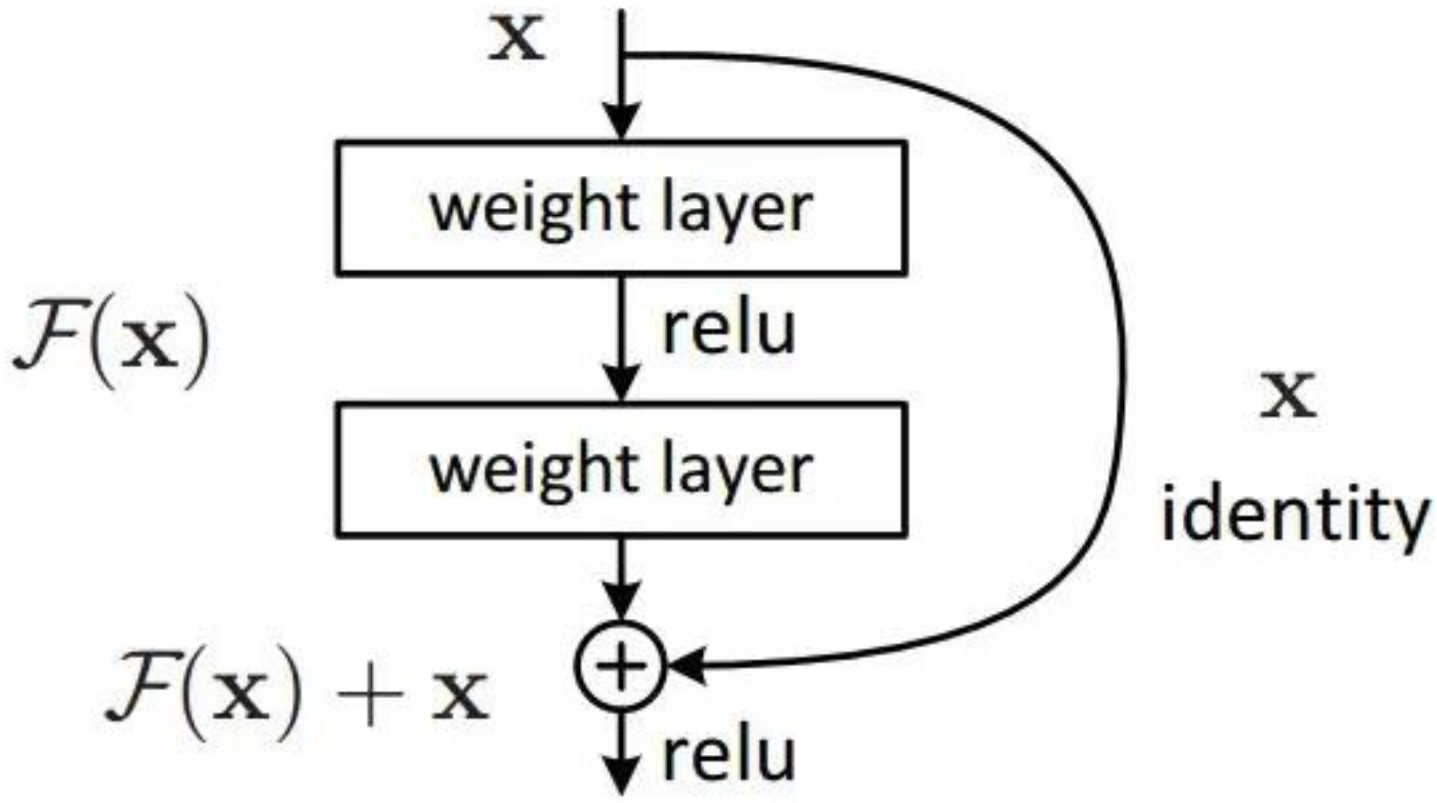
```
INPUT: [224x224x3]        memory:  224*224*3=150K   weights: 0
CONV3-64: [224x224x64]  memory:  224*224*64=3.2M   weights: (3*3*3)*64 = 1,728
CONV3-64: [224x224x64]  memory:  224*224*64=3.2M   weights: (3*3*64)*64 = 36,864
POOL2: [112x112x64]  memory:  112*112*64=800K   weights: 0
CONV3-128: [112x112x128]  memory:  112*112*128=1.6M   weights: (3*3*64)*128 = 73,728
CONV3-128: [112x112x128]  memory:  112*112*128=1.6M   weights: (3*3*128)*128 = 147,456
POOL2: [56x56x128]  memory:  56*56*128=400K   weights: 0
CONV3-256: [56x56x256]  memory:  56*56*256=800K   weights: (3*3*128)*256 = 294,912
CONV3-256: [56x56x256]  memory:  56*56*256=800K   weights: (3*3*256)*256 = 589,824
CONV3-256: [56x56x256]  memory:  56*56*256=800K   weights: (3*3*256)*256 = 589,824
POOL2: [28x28x256]  memory:  28*28*256=200K   weights: 0
CONV3-512: [28x28x512]  memory:  28*28*512=400K   weights: (3*3*256)*512 = 1,179,648
CONV3-512: [28x28x512]  memory:  28*28*512=400K   weights: (3*3*512)*512 = 2,359,296
CONV3-512: [28x28x512]  memory:  28*28*512=400K   weights: (3*3*512)*512 = 2,359,296
POOL2: [14x14x512]  memory:  14*14*512=100K   weights: 0
CONV3-512: [14x14x512]  memory:  14*14*512=100K   weights: (3*3*512)*512 = 2,359,296
CONV3-512: [14x14x512]  memory:  14*14*512=100K   weights: (3*3*512)*512 = 2,359,296
CONV3-512: [14x14x512]  memory:  14*14*512=100K   weights: (3*3*512)*512 = 2,359,296
POOL2: [7x7x512]  memory:  7*7*512=25K  weights: 0
FC: [1x1x4096]  memory:  4096  weights: 7*7*512*4096 = 102,760,448
FC: [1x1x4096]  memory:  4096  weights: 4096*4096 = 16,777,216
FC: [1x1x1000]  memory:  1000 weights: 4096*1000 = 4,096,000

TOTAL memory: 24M * 4 bytes ~= 93MB / image (only forward! ~*2 for bwd)
TOTAL params: 138M parameters
```

Residual Networks (ResNet)

- ResNet, also known as residual neural network, refers to the idea of adding residual learning to the traditional convolutional neural network,
- In order to solve the problem of the vanishing/exploding gradient, this architecture introduced the concept called Residual Blocks.
- In this network, we use a technique called skip connections. The skip connection connects activations of a layer to further layers by skipping some layers in between.
- This forms a residual block. Resnets are made by stacking these residual blocks together.

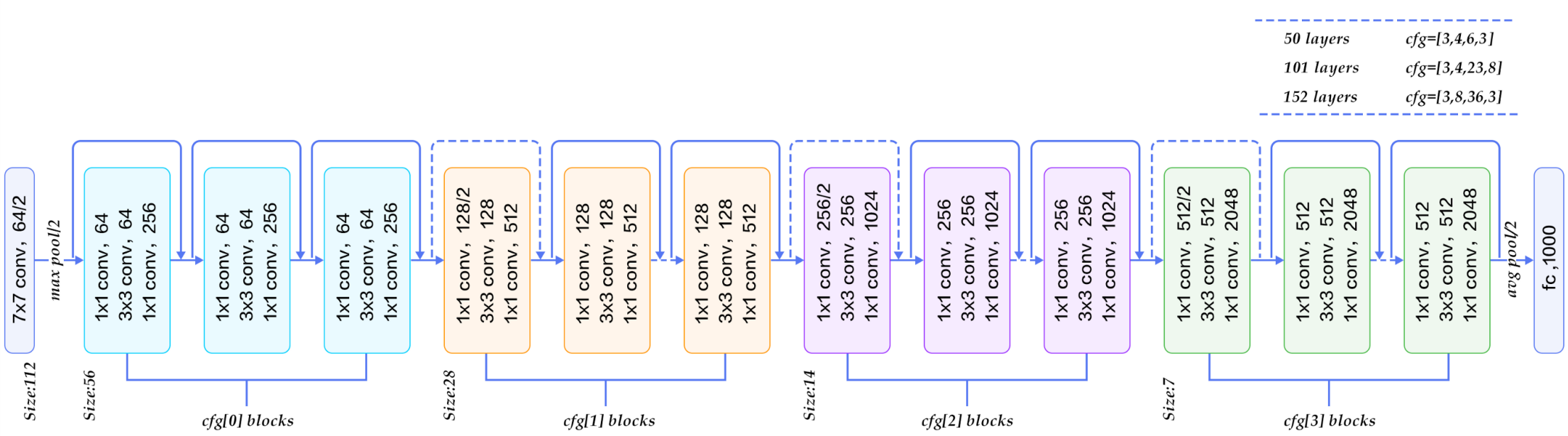


Residual Blocks



Residual Networks (ResNet)

ResNet



Transfer Learning and Pre-trained Models

- Transfer learning is a supervised learning method that aids construction of new models using pre-trained weights of previously constructed and fine-tuned models.

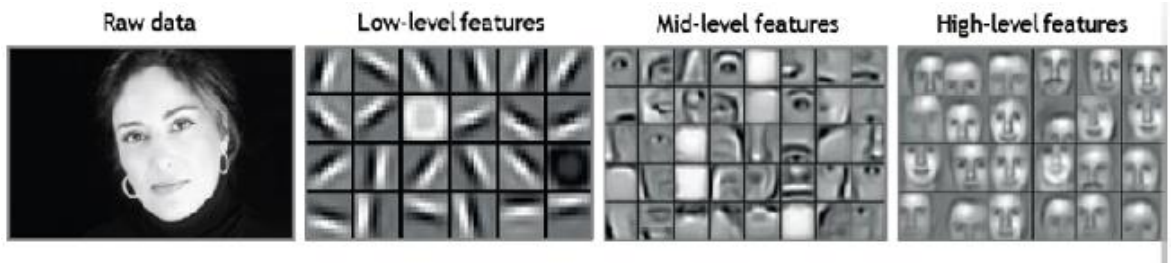
- Take a model trained on a large dataset and transfer its knowledge to a smaller dataset.

Unfreezing Layers

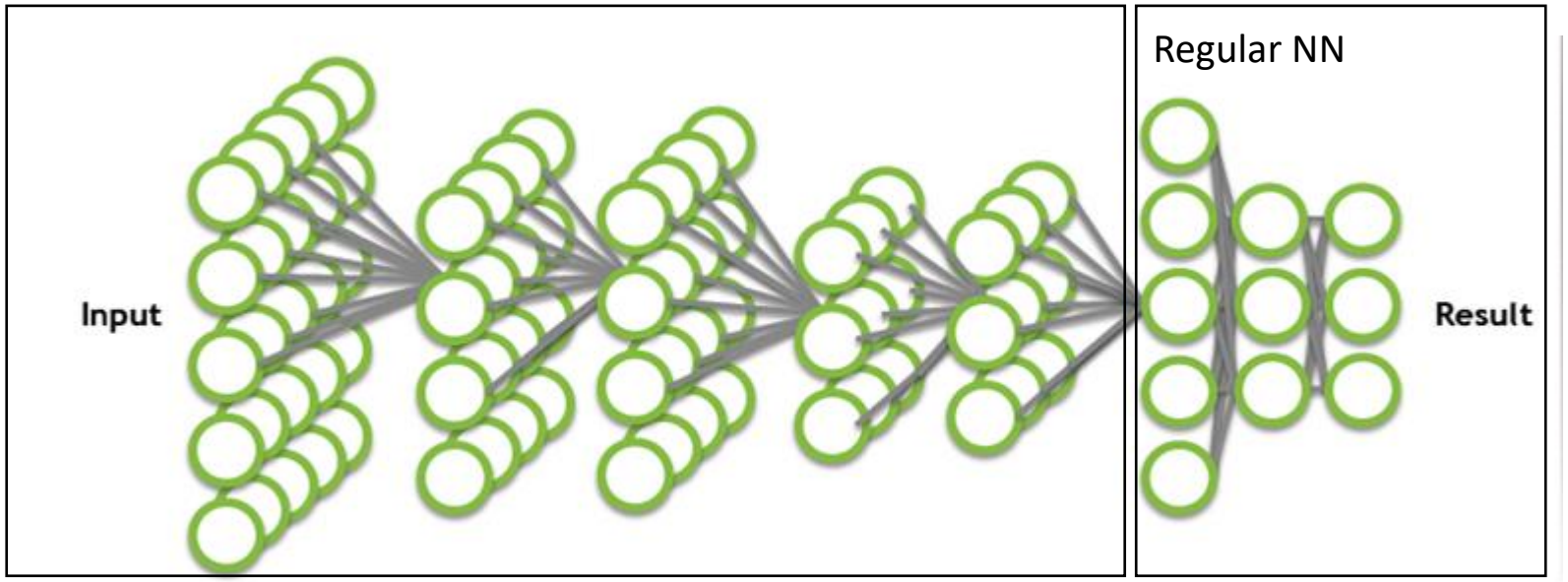
Source: https://www.tensorflow.org/tutorials/images/transfer_learning

- Unfreeze a few of the top layers of a frozen model base and jointly train both the newly-added classifier layers and the last layers of the base model.

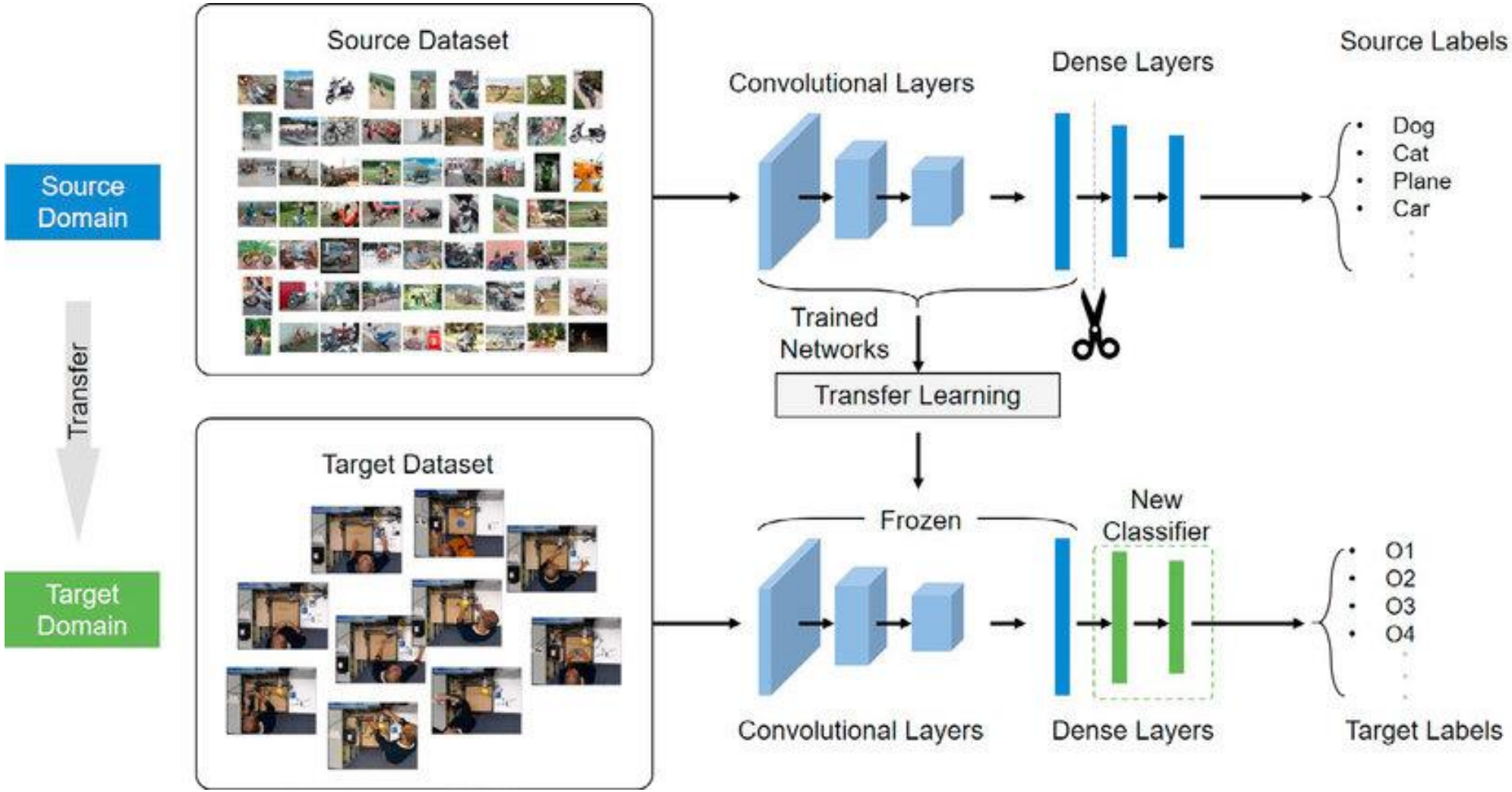
- This allows us to "fine-tune" the higher-order feature representations in the base model in order to make them more relevant for the specific task.



Output Label



Transfer Learning and Pre-trained Models



Transfer Learning and Pre-trained Models - Steps

1. Load in a pre-trained CNN model trained on a large dataset
2. Freeze parameters (weights) in model's lower convolutional layers
3. Add custom classifier with several layers of trainable parameters to model
4. Train classifier layers on training data available for task
5. Fine-tune hyperparameters and unfreeze more layers as needed

PyTorch Pre-trained Models

- https://pytorch.org/vision/stable/models.html

Available Pre-trained Models (Keras/ImageNet)

Source: https://keras.io/api/applications/

Model	Size (MB)	Top-1 Accuracy	Top-5 Accuracy	Parameters	Depth	Time (ms) per inference step (CPU)	Time (ms) per inference step (GPU)
Xception	88	0.790	0.945	22,910,480	126	109.42	8.06
VGG16	528	0.713	0.901	138,357,544	23	69.50	4.16
VGG19	549	0.713	0.900	143,667,240	26	84.75	4.38
ResNet50	98	0.749	0.921	25,636,712	-	58.20	4.55
ResNet101	171	0.764	0.928	44,707,176	-	89.59	5.19
ResNet152	232	0.766	0.931	60,419,944	-	127.43	6.54
ResNet50V2	98	0.760	0.930	25,613,800	-	45.63	4.42
ResNet101V2	171	0.772	0.938	44,675,560	-	72.73	5.43
ResNet152V2	232	0.780	0.942	60,380,648	-	107.50	6.64
InceptionV3	92	0.779	0.937	23,851,784	159	42.25	6.86
InceptionResNetV2	215	0.803	0.953	55,873,736	572	130.19	10.02
MobileNet	16	0.704	0.895	4,253,864	88	22.60	3.44

Available Pre-trained Models (Keras/ImageNet)

Source: https://keras.io/api/applications/

Model	Size (MB)	Top-1 Accuracy	Top-5 Accuracy	Parameters	Depth	Time (ms) per inference step (CPU)	Time (ms) per inference step (GPU)
MobileNetV2	14	0.713	0.901	3,538,984	88	25.90	3.83
DenseNet121	33	0.750	0.923	8,062,504	121	77.14	5.38
DenseNet169	57	0.762	0.932	14,307,880	169	96.40	6.28
DenseNet201	80	0.773	0.936	20,242,984	201	127.24	6.67
NASNetMobile	23	0.744	0.919	5,326,716	-	27.04	6.70
NASNetLarge	343	0.825	0.960	88,949,818	-	344.51	19.96
EfficientNetB0	29	-	-	5,330,571	-	46.00	4.91
EfficientNetB1	31	-	-	7,856,239	-	60.20	5.55
EfficientNetB2	36	-	-	9,177,569	-	80.79	6.50
EfficientNetB3	48	-	-	12,320,535	-	139.97	8.77
EfficientNetB4	75	-	-	19,466,823	-	308.33	15.12
EfficientNetB5	118	-	-	30,562,527	-	579.18	25.29
EfficientNetB6	166	-	-	43,265,143	-	958.12	40.45

Pre-trained VGG Example (Tensorflow Keras)

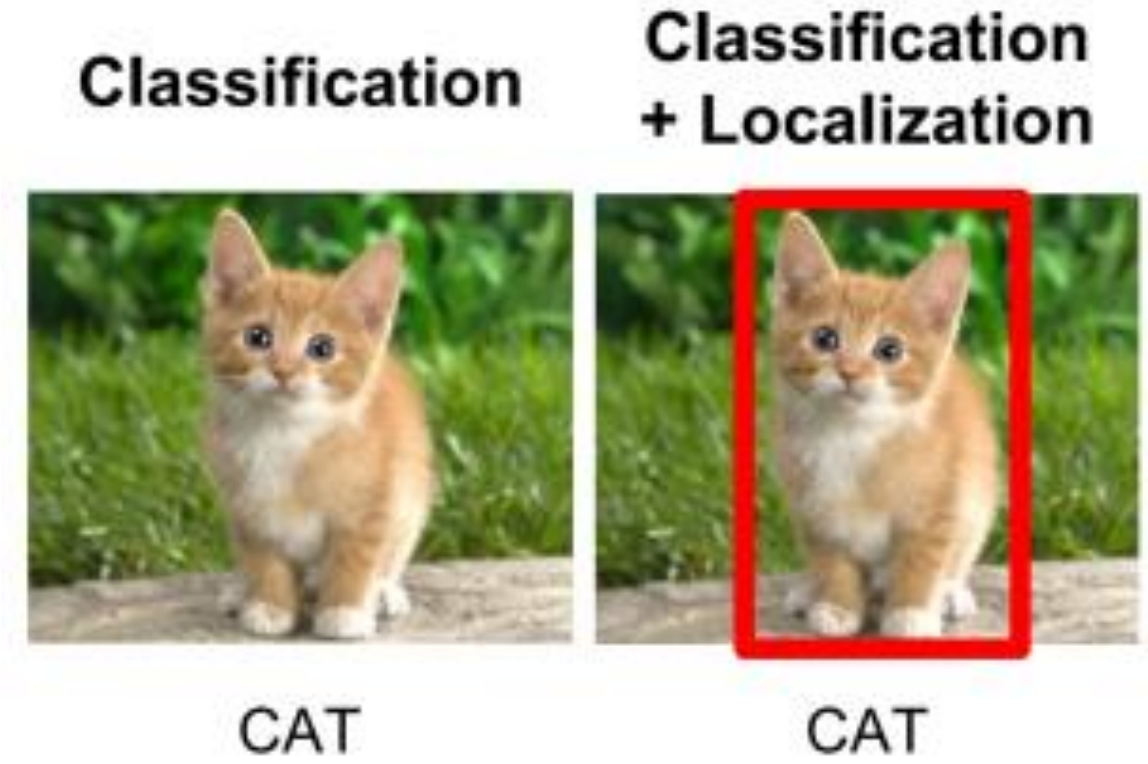
- Classification of Fashion Images
- https://www.kaggle.com/code/anandad/classify-fashion-mnist-with-vgg16

Pre-trained ResNet Example (PyTorch)

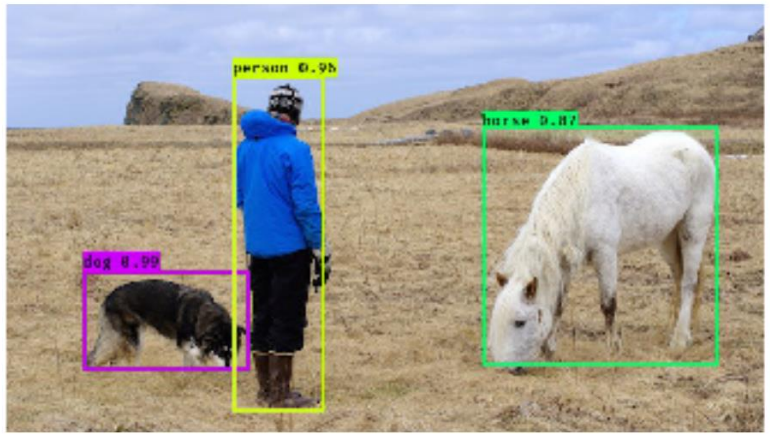
- Classification of Weather Images using ResNet-34 in PyTorch
  - https://github.com/chiragdaryani/Weather-Images-Classification-in-PyTorch

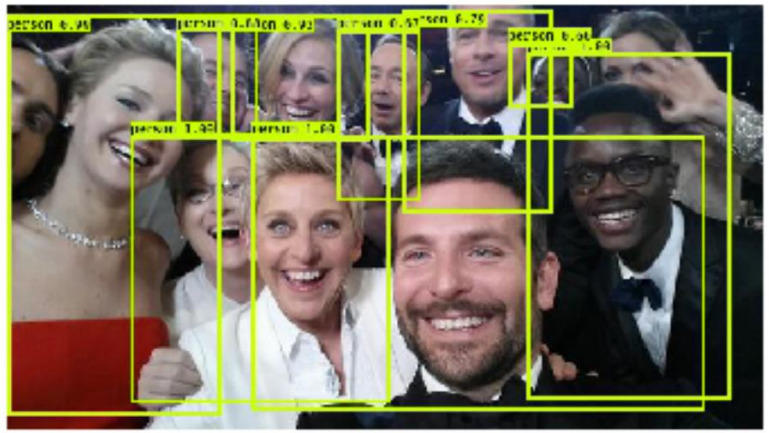
Other Computer Vision Applications

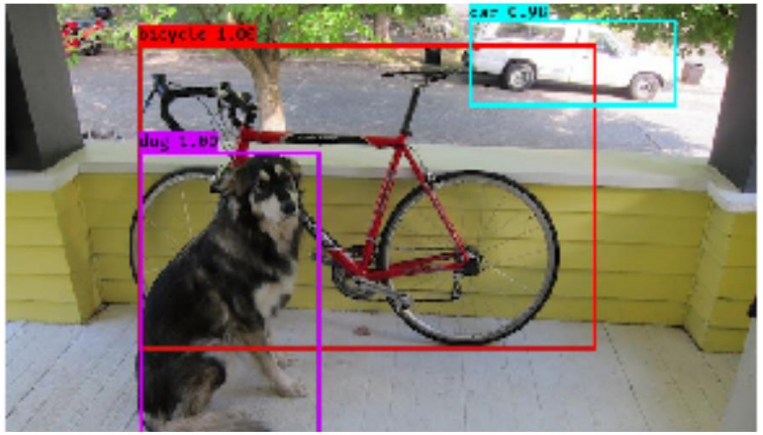
Image Classification with Localization

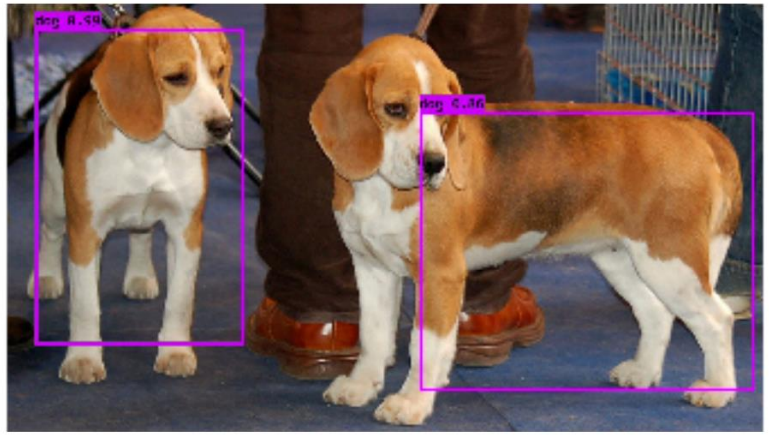


Object Detection

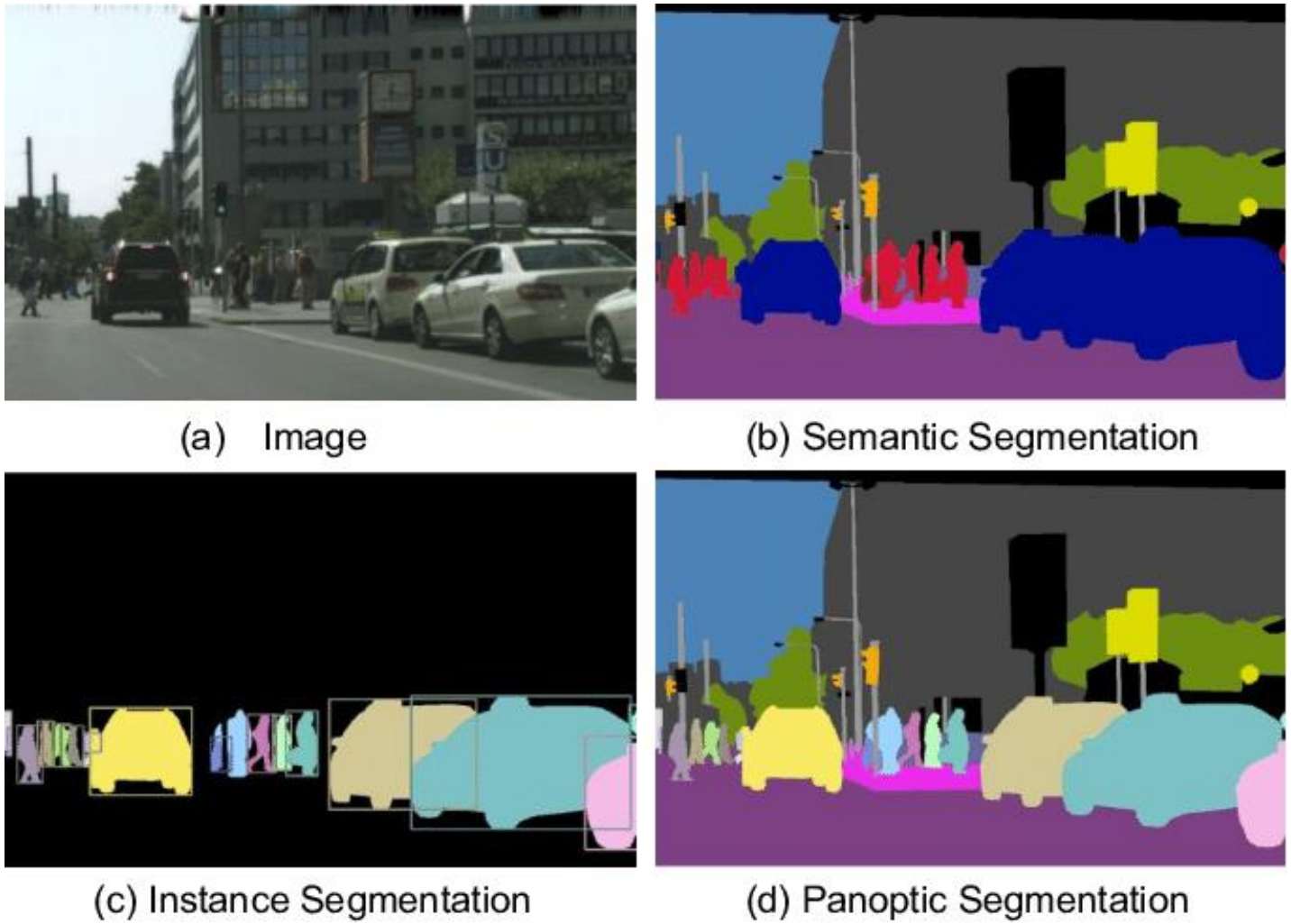








Semantic, Instance and Panoptic Segmentation



Keypoint (Object Landmark) Detection

Object Tracking

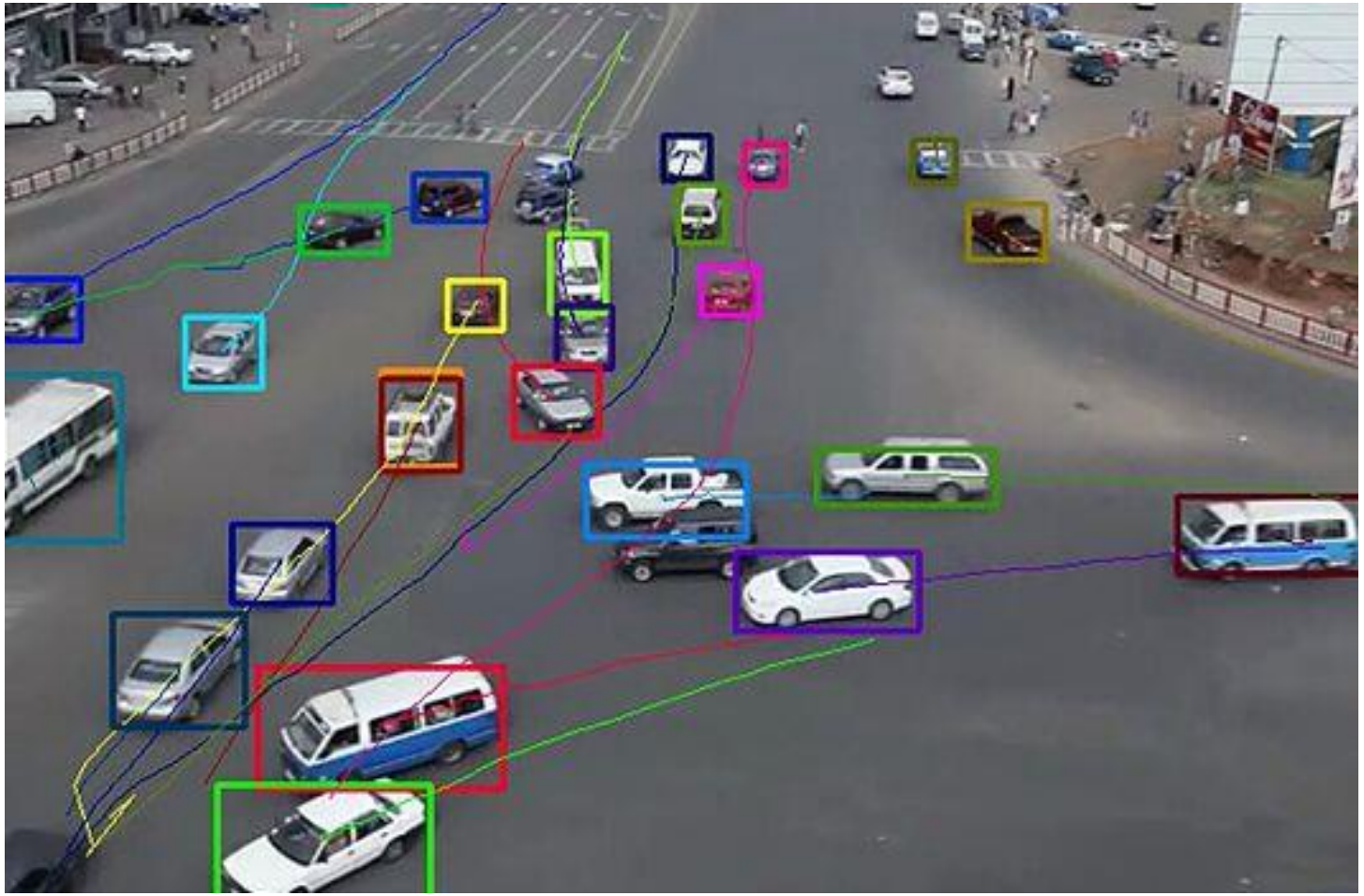
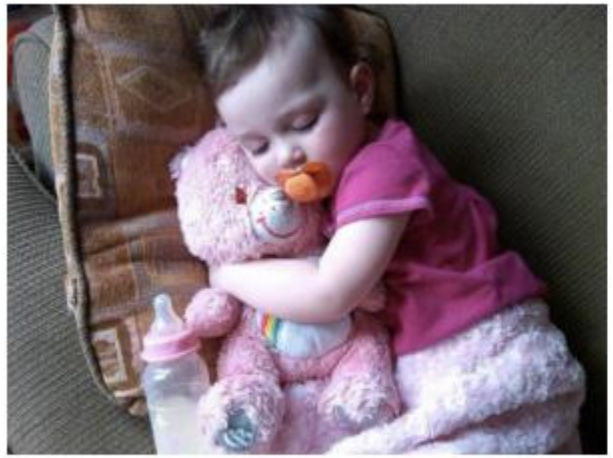


Image Captioning



A close up of a child holding a stuffed animal

Two pizzas sitting on top of a stove top oven

A man flying through the air while riding a skateboard

Text to Image Generation

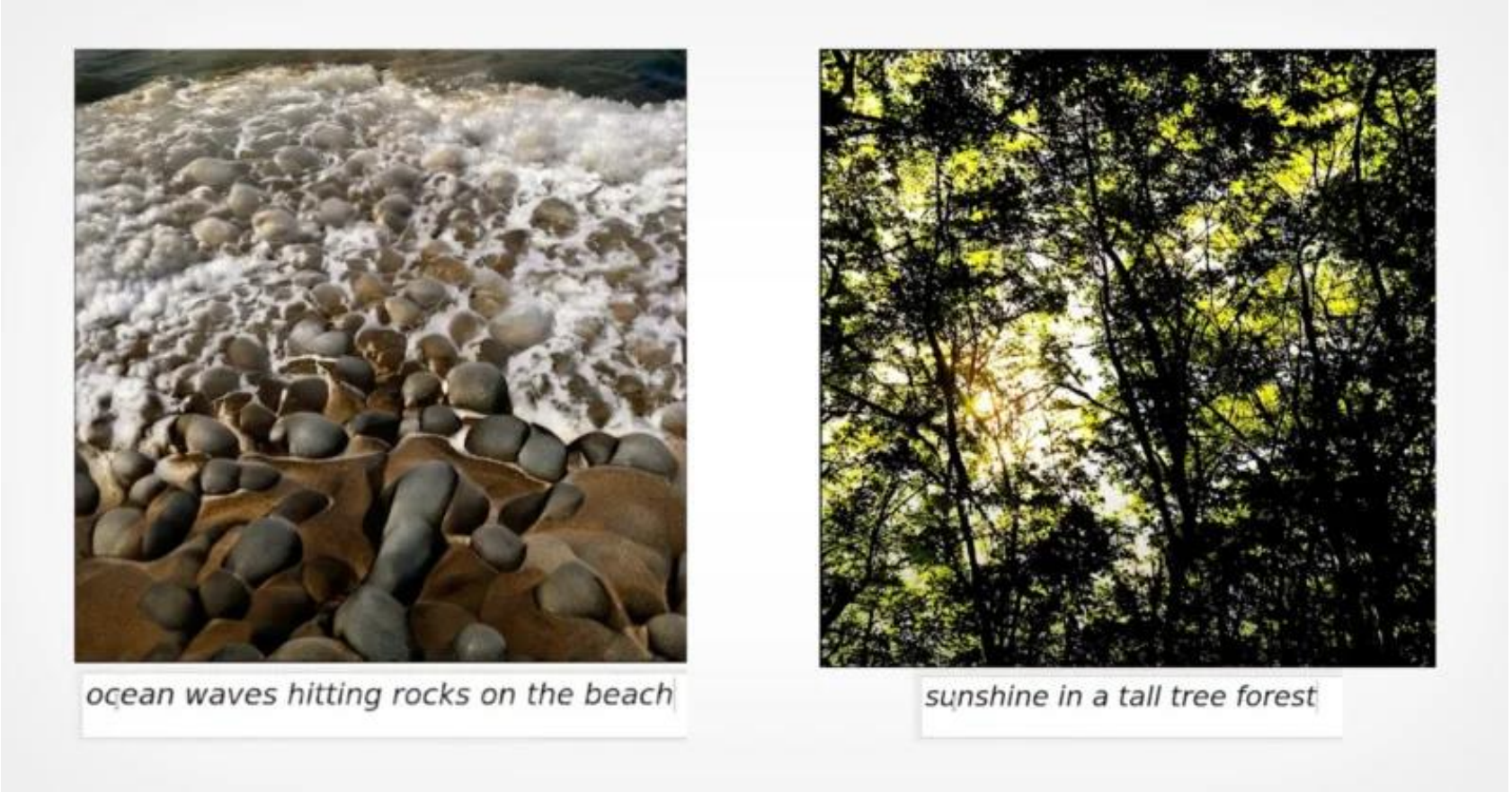
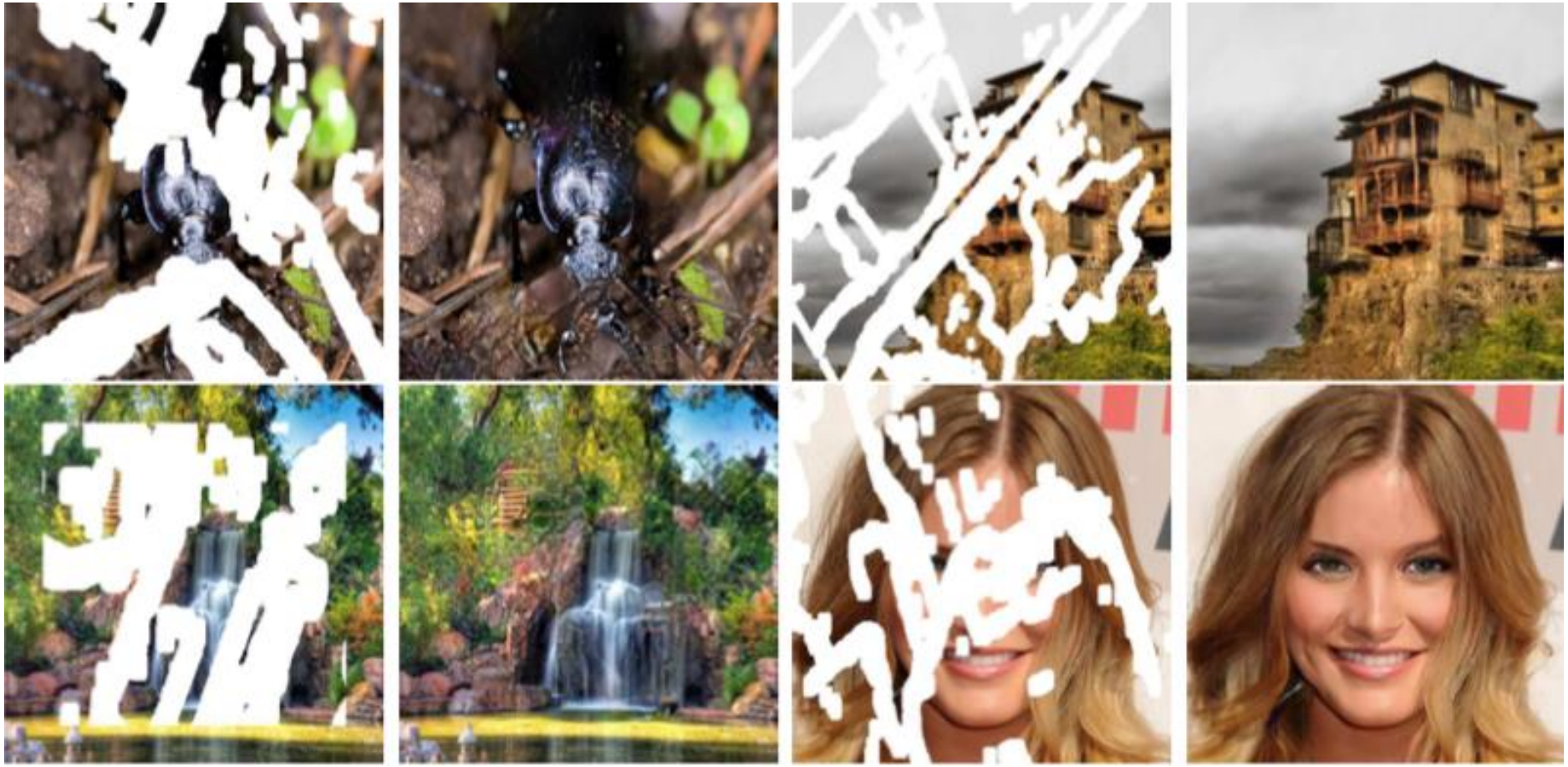


Image Reconstruction



Deep Learning Models for Computer Vision

Image Classification

- AlexNet
- GoogleNet
- VGGNet
- ResNet
- Inception
- Xception
- MobileNet
- DenseNet

Object Detection

- Fast R-CNN
- Faster R-CNN
- YOLO
- SSD

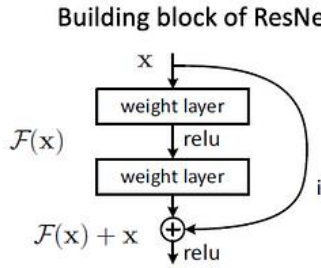
Instance segmentation

Mask R-CNN

Keypoint Detection

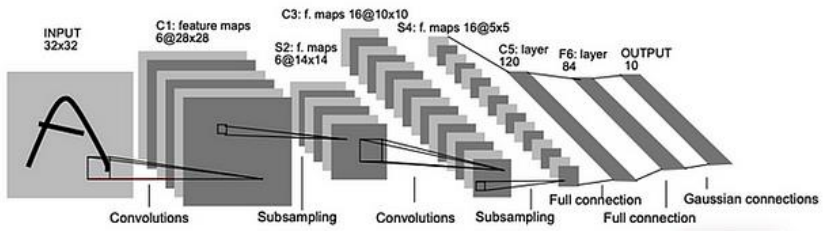
Keypoint R-CNN

Computer Vision Milestones

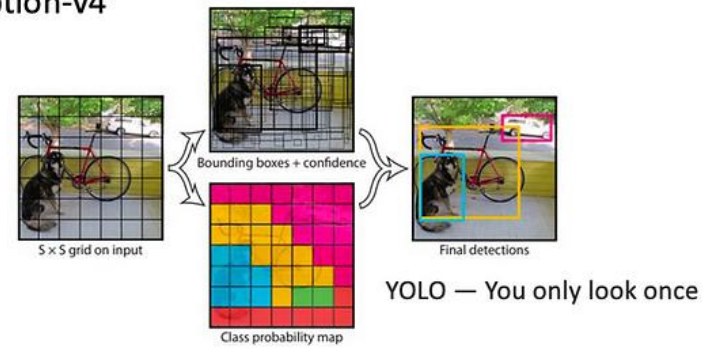


Viola-Jones 2001
HoG 2006
ImageNet 2009
AlexNet (deepCNN) 2012
Fast R-CNN
ResNet 2015
FPN
Mask R-CNN
RevNet
YOLOv2
Transformer 2017
Vision Transformer (ViT) 2021

1998 LeNet
2010 Pascal VOC
2014 VGG
R-CNN
GoogLeNet (Inception)
2016 SSD
YOLO
Inception-v4
2018 YOLOv3
BERT
2020 Image GPT
DETR
2022 YOLOv7



The architecture of LeNet-5 for digit recognition by Yann LeCun



YOLO — You only look once

Bullets

Summary of Techniques for Computer Vision Problems

- Convolutional feature maps
- Region proposals
- Two-stage and one-stage detectors
- Bounding box regressors
- Vision transformers (self and multi-head attention)
- Non-maximum suppression(NMS)
- Anchor boxes
- Instance segmentation
- Augmentation
- Pseudo labeling
- Self-supervised learning
- Panoptic Segmentation for assign pixel level annotation