Reconocimiento de Patrones

Version 2023-2

Deep Learning

[Capítulo 4]

Dr. José Ramón Iglesias
DSP-ASIC BUILDER GROUP
Director Semillero TRIAC
Ingenieria Electronica
Universidad Popular del Cesar

npj | Digital Medicine

www.nature.com/npjdigitalmed

Corrected: Publisher Correction

ARTICLE OPEN

Development and validation of a deep learning algorithm for improving Gleason scoring of prostate cancer

Kunal Nagpal[1], Davis Foote[1], Yun Liu[1], Po-Hsuan Cameron Chen[1], Ellery Wulczyn[1], Fraser Tan[1], Niels Olson[2], Jenny L. Smith[2], Arash Mohtashamian[2], James H. Wren[3], Greg S. Corrado[1], Robert MacDonald[1], Lily H. Peng[1], Mahul B. Amin[4], Andrew J. Evans[5], Ankur R. Sangoi[6], Craig H. Mermel[1], Jason D. Hipp[1] and Martin C. Stumpe[1,7]

For prostate cancer patients, the Gleason score is one of the most important prognostic factors, potentially determining treatment independent of the stage. However, Gleason scoring is based on subjective microscopic examination of tumor morphology and suffers from poor reproducibility. Here we present a deep learning system (DLS) for Gleason scoring whole-slide images of prostatectomies. Our system was developed using 112 million pathologist-annotated image patches from 1226 slides, and evaluated on an independent validation dataset of 331 slides. Compared to a reference standard provided by genitourinary pathology experts, the mean accuracy among 29 general pathologists was 0.61 on the validation set. The DLS achieved a significantly higher diagnostic accuracy of 0.70 ($p = 0.002$) and trended towards better patient risk stratification in correlations to clinical follow-up data. Our approach could improve the accuracy of Gleason scoring and subsequent therapy decisions, particularly where specialist expertise is unavailable. The DLS also goes beyond the current Gleason system to more finely characterize and quantitate tumor morphology, providing opportunities for refinement of the Gleason system itself.

npj Digital Medicine (2019)2:48 ; https://doi.org/10.1038/s41746-019-0112-2

https://www.nature.com/articles/s41746-019-0112-2.pdf

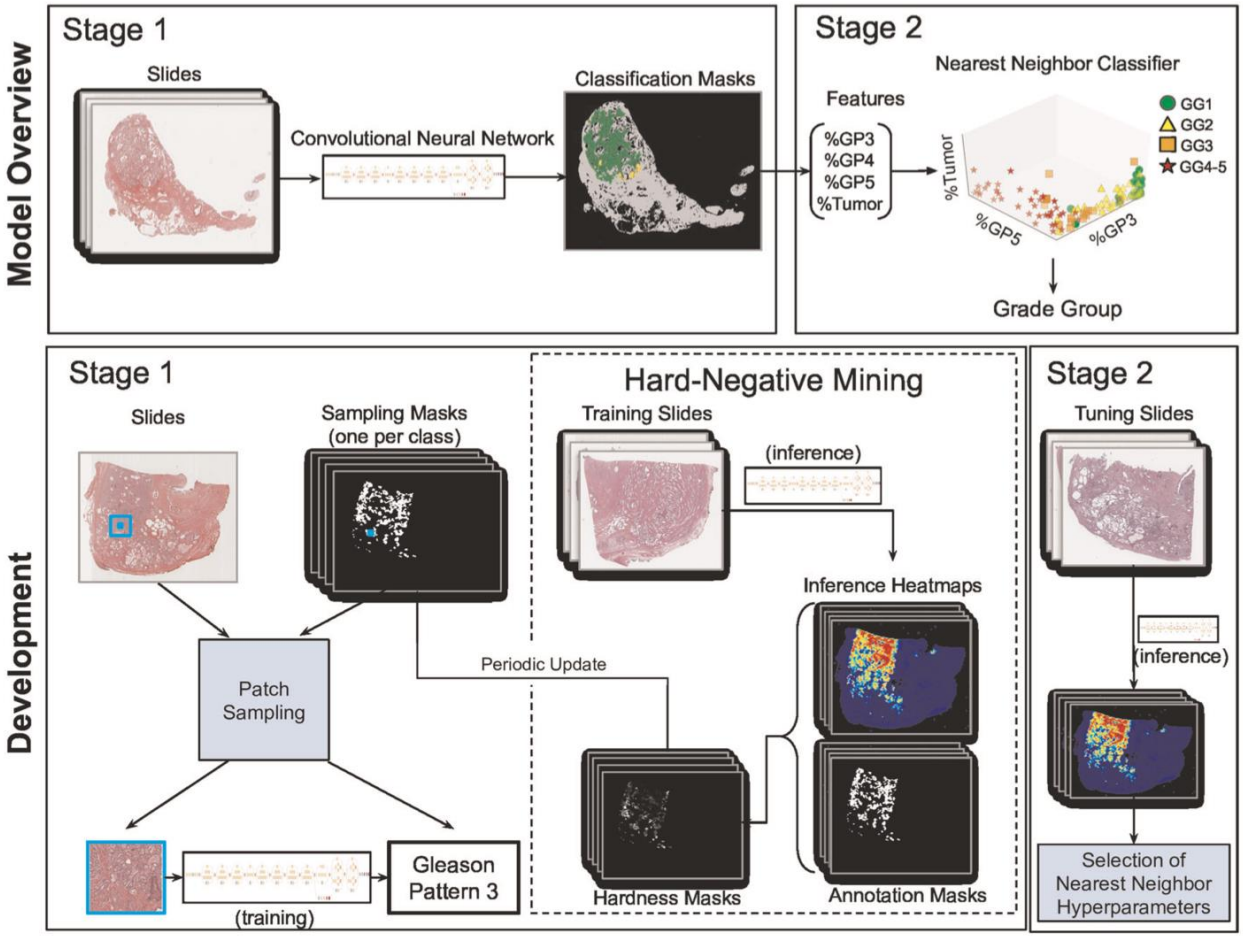


Fig. 1 Illustration of the development and usage of the two-stage deep learning system (DLS). Developing the DLS involves training two machine learning models. Stage 1 is an ensembled deep convolutional neural network (CNN) that classifies every region in the slide as non-tumor or its Gleason pattern (GP). Training the stage 1 CNN involves first collecting pathologists' annotations (Annotation Masks) of whole-slide images at the region level, and then generating "sampling masks" indicating the locations of each of the four classes (non-tumor, GP3, GP4, and GP5) for each slide. Over the course of millions of training iterations, sampled image patches and associated labels are used to train the constituent CNNs in the ensembled stage 1 CNN model. During the training process, we performed hard-negative mining by periodically applying each individual partially trained model to the entire training corpus of whole-slide images. Comparison of these intermediate inference results to the original annotations highlights the most difficult image patches, and we focus training on these patches. Stage 2 involves first collecting pathologists' labels of the Gleason Grade Group (GG) for each slide. Next, the predictions of the stage 1 model are calibrated and converted to four features that indicate the amount of tumor and each GP in the slide. k-nearest-neighbor (kNN) classifiers are then trained to predict the GG (1, 2, 3, or 4–5), or whether the GG is above specific thresholds (GG ≥ 2, GG ≥ 3, or GG ≥ 4). For more details, please refer to the "Deep Learning System" section in the Supplement

LETTER

doi:10.1038/nature21056

Dermatologist-level classification of skin cancer with deep neural networks

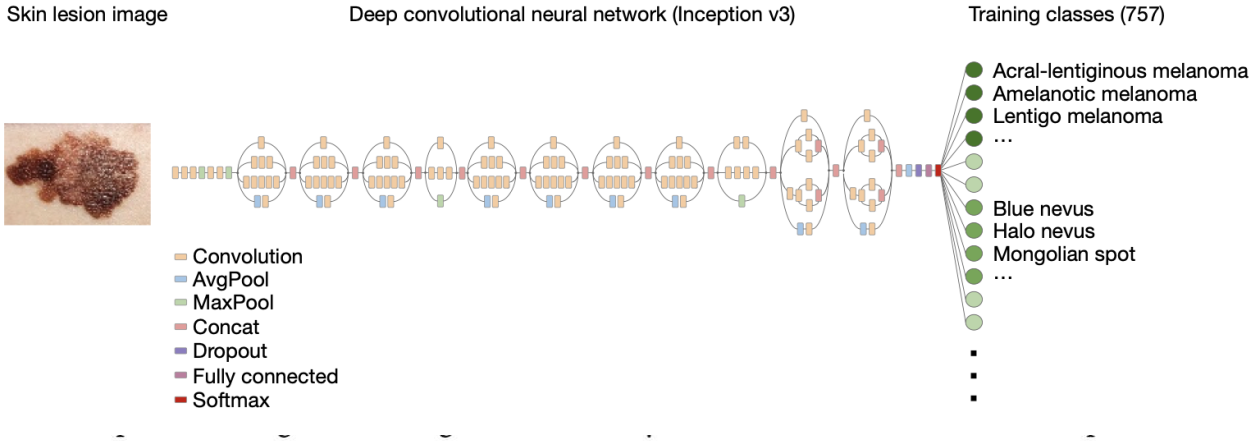
Andre Esteva[1]*, Brett Kuprel[1]*, Roberto A. Novoa[2,3], Justin Ko[2], Susan M. Swetter[2,4], Helen M. Blau[5] & Sebastian Thrun[6]

Skin cancer, the most common human malignancy[1–3], is primarily diagnosed visually, beginning with an initial clinical screening and followed potentially by dermoscopic analysis, a biopsy and histopathological examination. Automated classification of skin lesions using images is a challenging task owing to the fine-grained variability in the appearance of skin lesions. Deep convolutional neural networks (CNNs)[4,5] show potential for general and highly variable tasks across many fine-grained object categories[6–11]. Here we demonstrate classification of skin lesions using a single CNN, trained end-to-end from images directly, using only pixels and disease labels as inputs. We train a CNN using a dataset of 129,450 clinical images—two orders of magnitude larger than previous datasets[12]—consisting of 2,032 different diseases. We

images (for example, smartphone images) exhibit variability in factors such as zoom, angle and lighting, making classification substantially more challenging[23,24]. We overcome this challenge by using a data-driven approach—1.41 million pre-training and training images make classification robust to photographic variability. Many previous techniques require extensive preprocessing, lesion segmentation and extraction of domain-specific visual features before classification. By contrast, our system requires no hand-crafted features; it is trained end-to-end directly from image labels and raw pixels, with a single network for both photographic and dermoscopic images. The existing body of work uses small datasets of typically less than a thousand images of skin lesions[16,18,19], which, as a result, do not generalize well to new images. We demonstrate generalizable classification with a new

https://www.nature.com/articles/nature21056.pdf

Skin Cancer Recognition



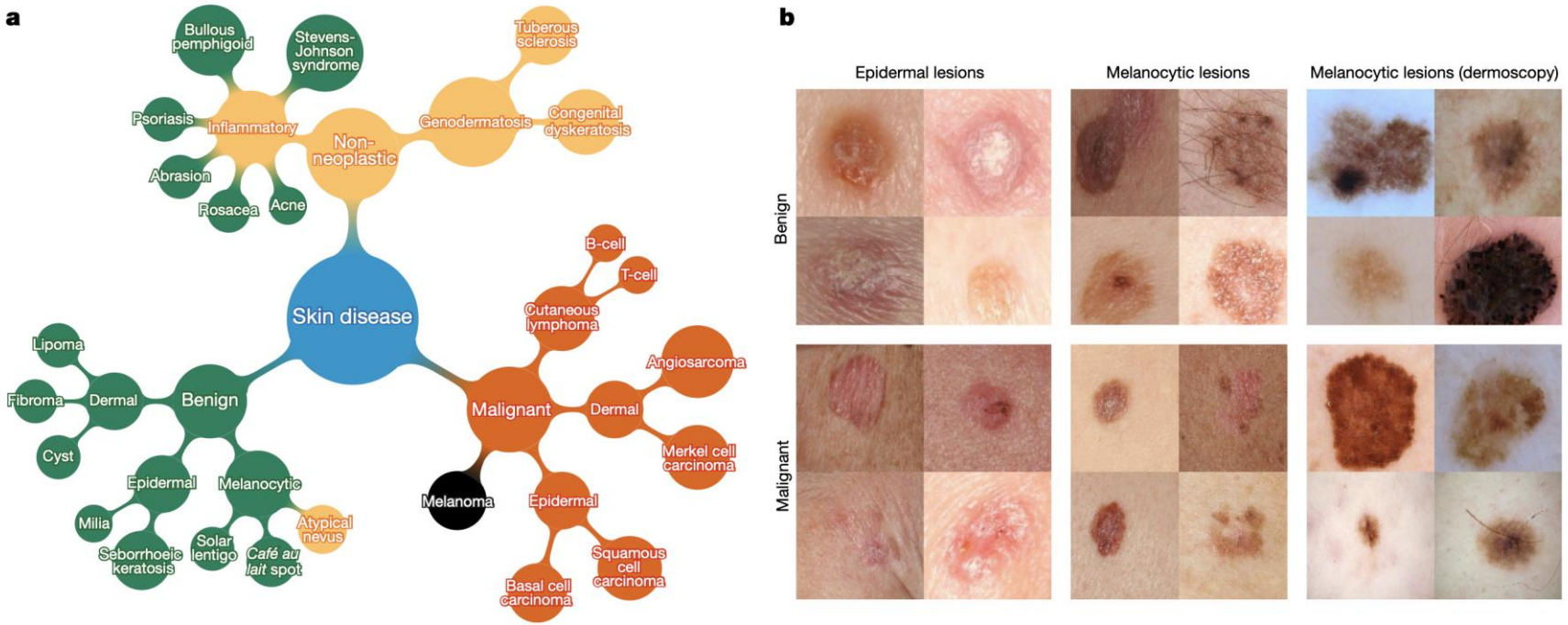


Figure 2 | A schematic illustration of the taxonomy and example test set images. a, A subset of the top of the tree-structured taxonomy of skin disease. The full taxonomy contains 2,032 diseases and is organized based on visual and clinical similarity of diseases. Red indicates malignant, green indicates benign, and orange indicates conditions that can be either. Black indicates melanoma. The first two levels of the taxonomy are used in validation. Testing is restricted to the tasks of **b**. **b**, Malignant and benign example images from two disease classes. These test images highlight the difficulty of malignant versus benign discernment for the three medically critical classification tasks we consider: epidermal lesions, melanocytic lesions and melanocytic lesions visualized with a dermoscope. Example images reprinted with permission from the Edinburgh Dermofit Library (https://licensing.eri.ed.ac.uk/i/software/dermofit-image-library.html).

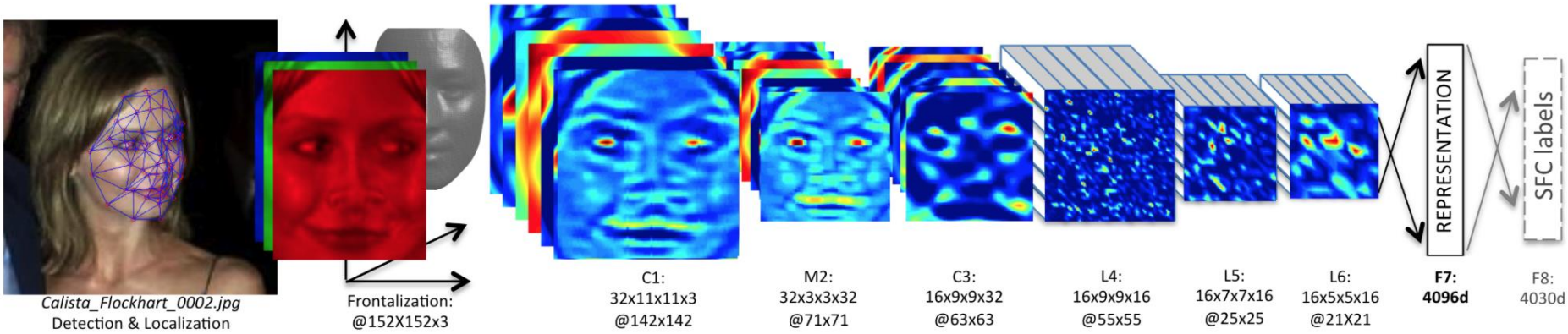


Figure 2. **Outline of the *DeepFace* architecture**. A front-end of a single convolution-pooling-convolution filtering on the rectified input, followed by three locally-connected layers and two fully-connected layers. Colors illustrate feature maps produced at each layer. The net includes more than 120 million parameters, where more than 95% come from the local and fully connected layers.

Taigman, Yaniv, et al. "Deepface: Closing the gap to human-level performance in face verification." *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2014. [PDF]

OverView Data Discussion Results Submits

Challenge 3: Face Feature Test/Trillion Pairs

Chrome browser is strongly recommended for normal displaying of our website.

Participation

Register and download our training and testing data sets.

You can submit the features of your model and you will get all results in less than 1-2 hours.

You can post your questions in Discussion page.

Datasets

- Training set
    1. MS-Celeb-1M-v1c with 86,876 ids/3,923,399 aligned images cleaned from MS-Celeb-1M dataset. This dataset has been excluded from both LFW and Asian-Celeb.
    2. Asian-Celeb 93,979 ids/2,830,146 aligned images. This dataset has been excluded from both LFW and MS-Celeb-1M-v1c.
- Testing set

  Trillion Pairs

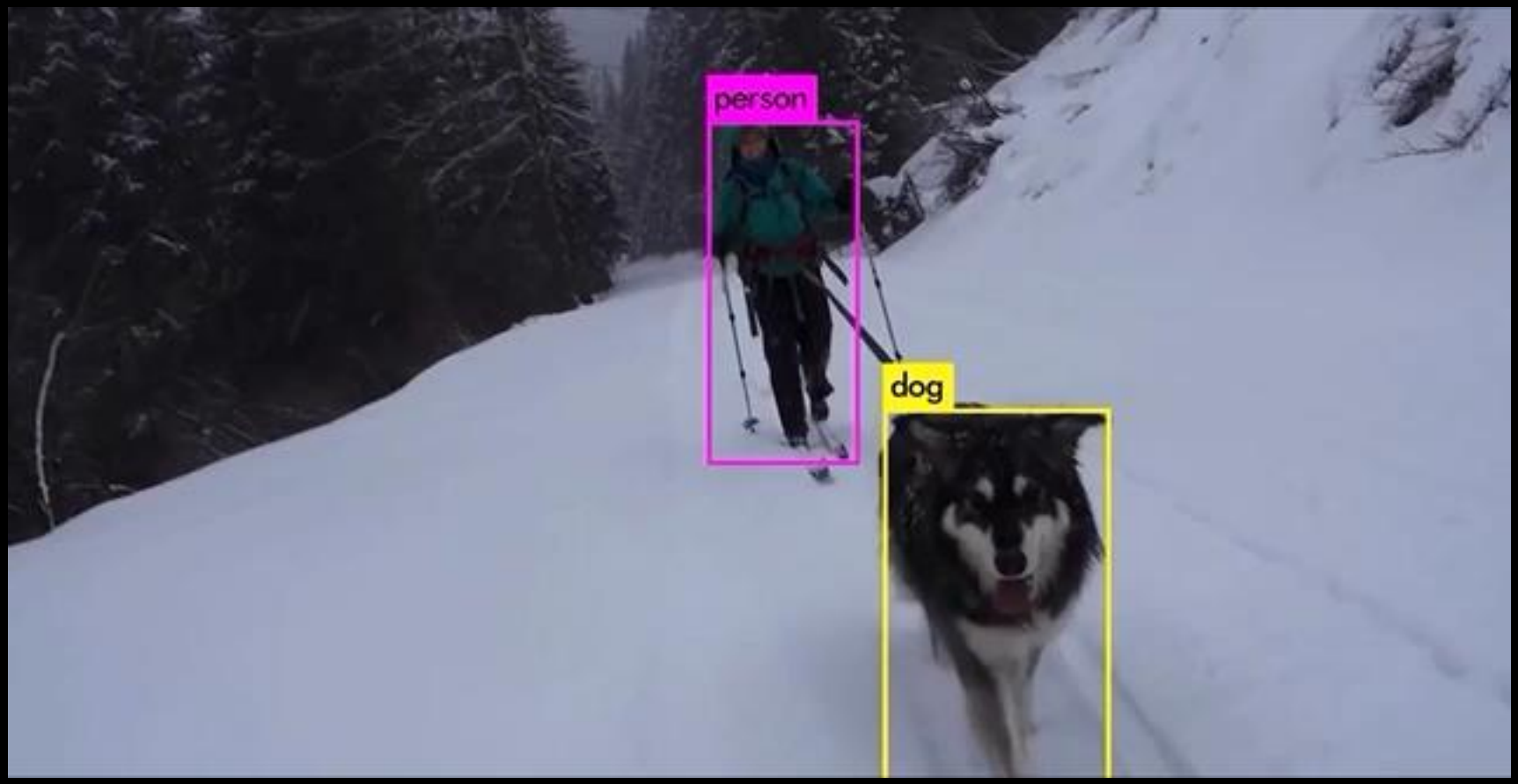
  Trillion Pairs is consisted of the following two parts.

    1. ELFW: Face images of celebrities in LFW name list. There are 274k images from 5.7k ids.
    2. DELFW: Distractors for ELFW. There are in total 1.58 million face images from Flickr.

Fig. 9 Some example images with object annotations from PASCAL VOC, ILSVRC, MS COCO and Open Images. See Table 2 for a summary of these datasets

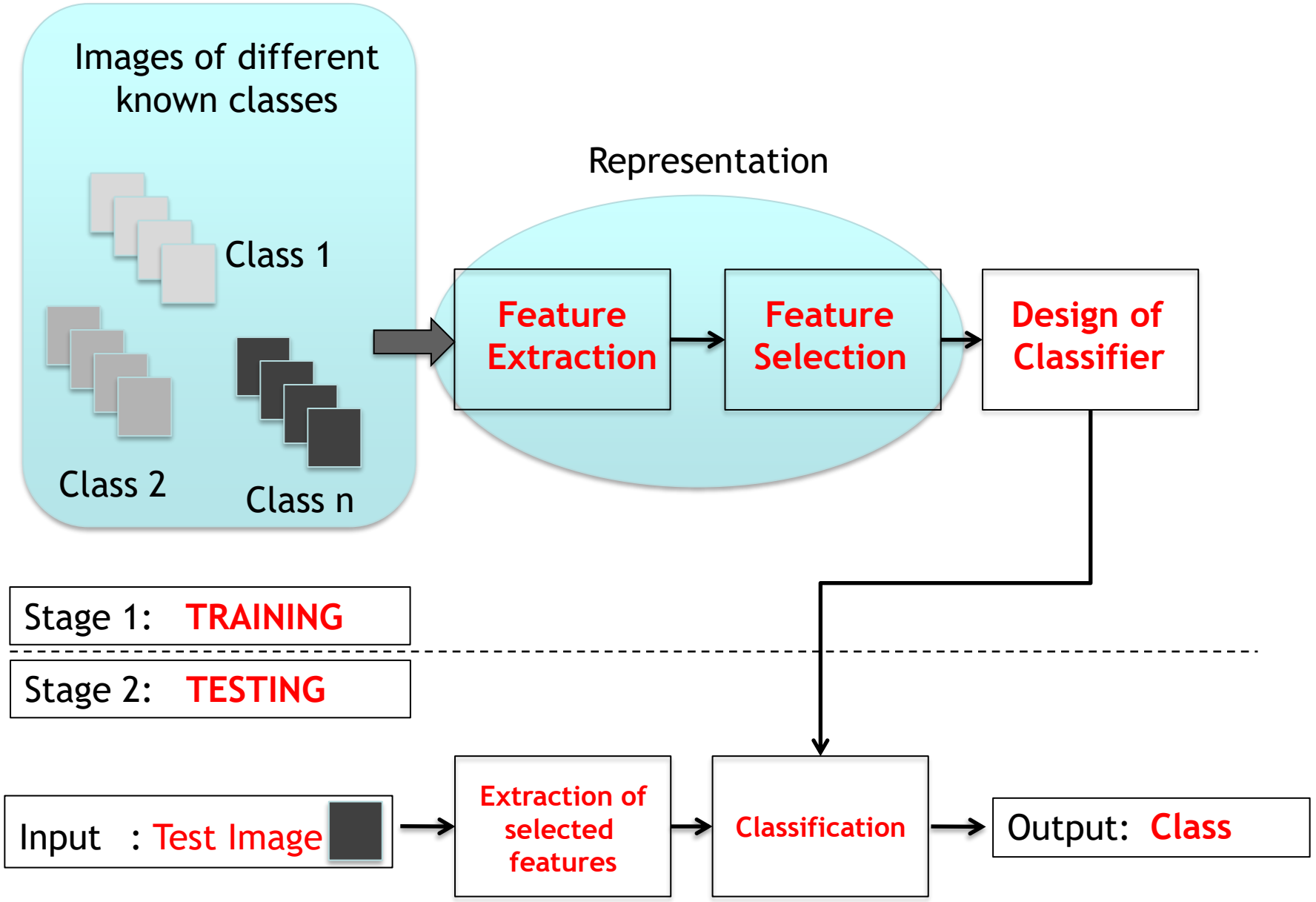
Liu, L., Ouyang, W., Wang, X., Fieguth, P., Chen, J., Liu, X., & Pietikäinen, M. (2020). Deep learning for generic object detection: A survey. *International journal of computer vision*, *128*(2), 261-318.

Object Detection



https://pjreddie.com/darknet/yolo/

Motivation


Images of different known classes
Class 1
Class 2
Class n
Representation
Feature Extraction
Feature Selection
Design of Classifier
Stage 1: TRAINING
Stage 2: TESTING
Input : Test Image
Extraction of selected features
Classification
Output: Class

MOTIVATION

- Handcrafted vs. Learned features

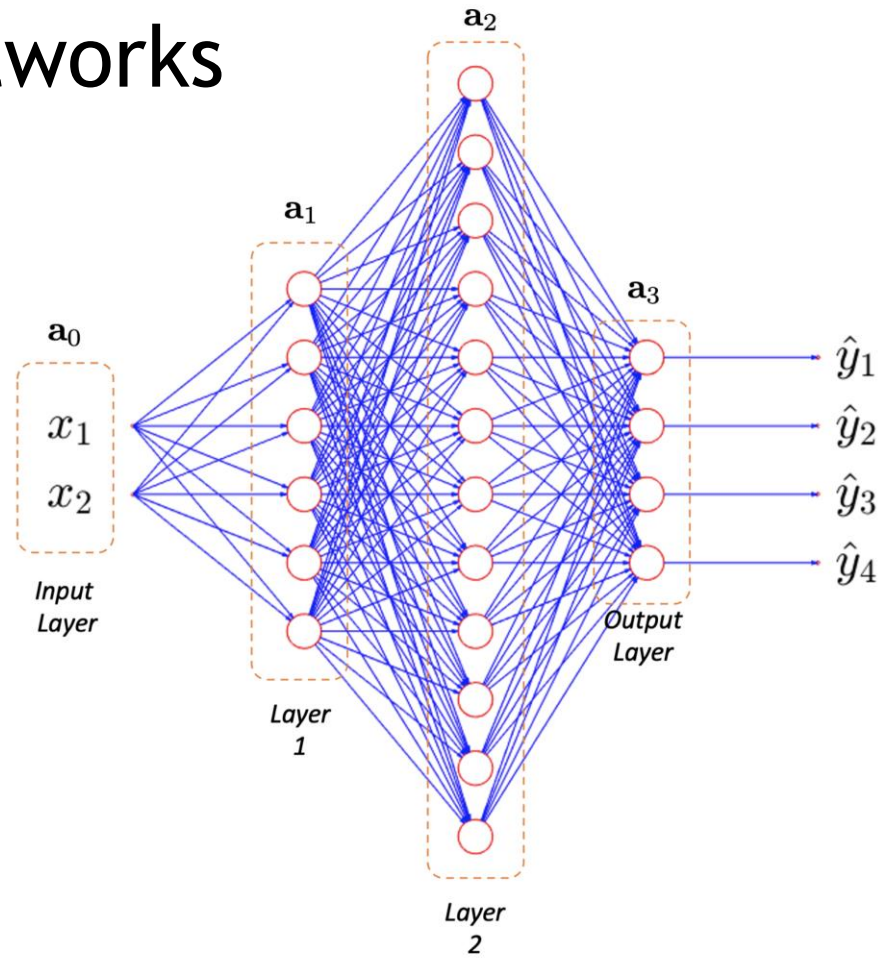
LBP, Haralick, Fourier, Gabor, etc.

or

Learned representation from training data

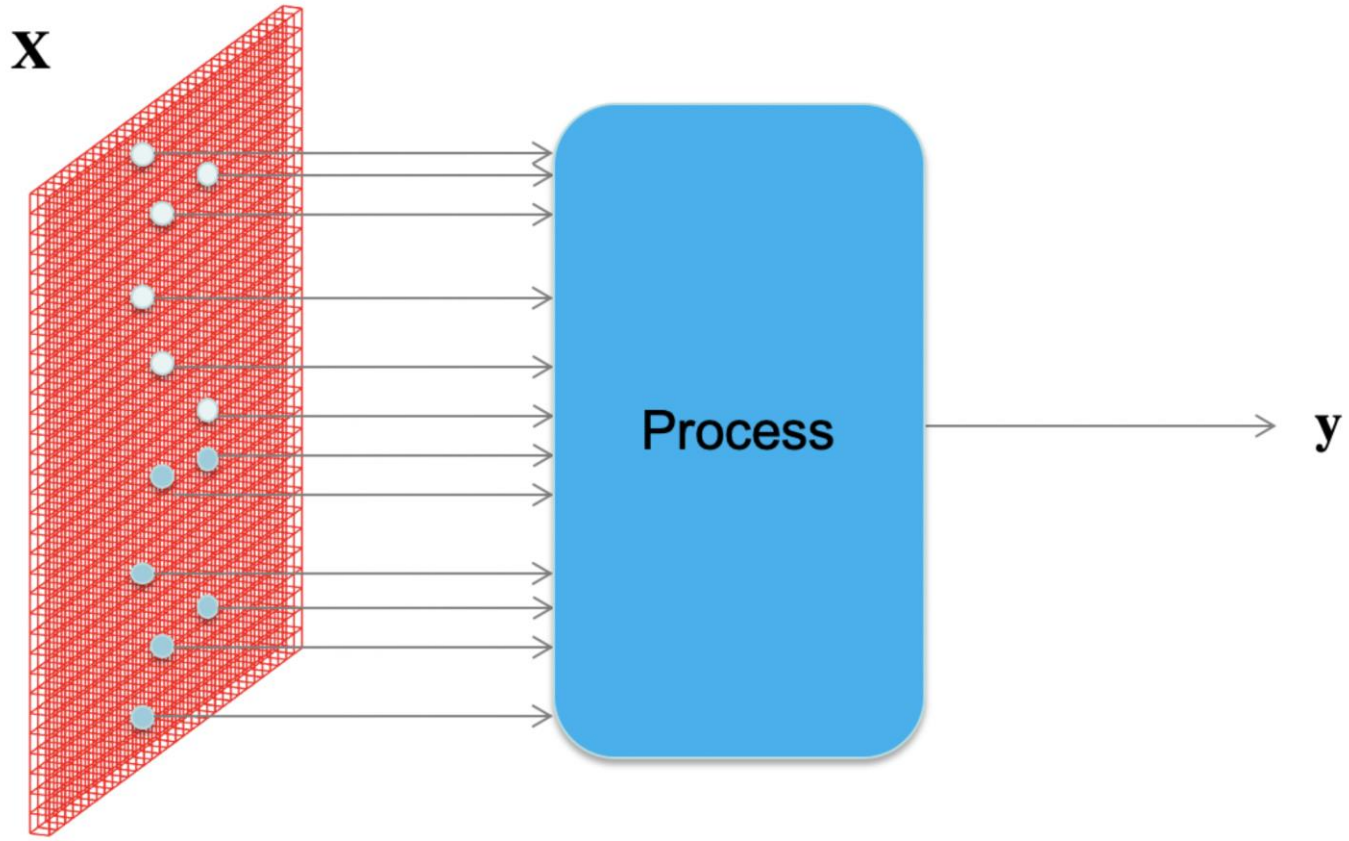
MOTIVATION

- Neural networks



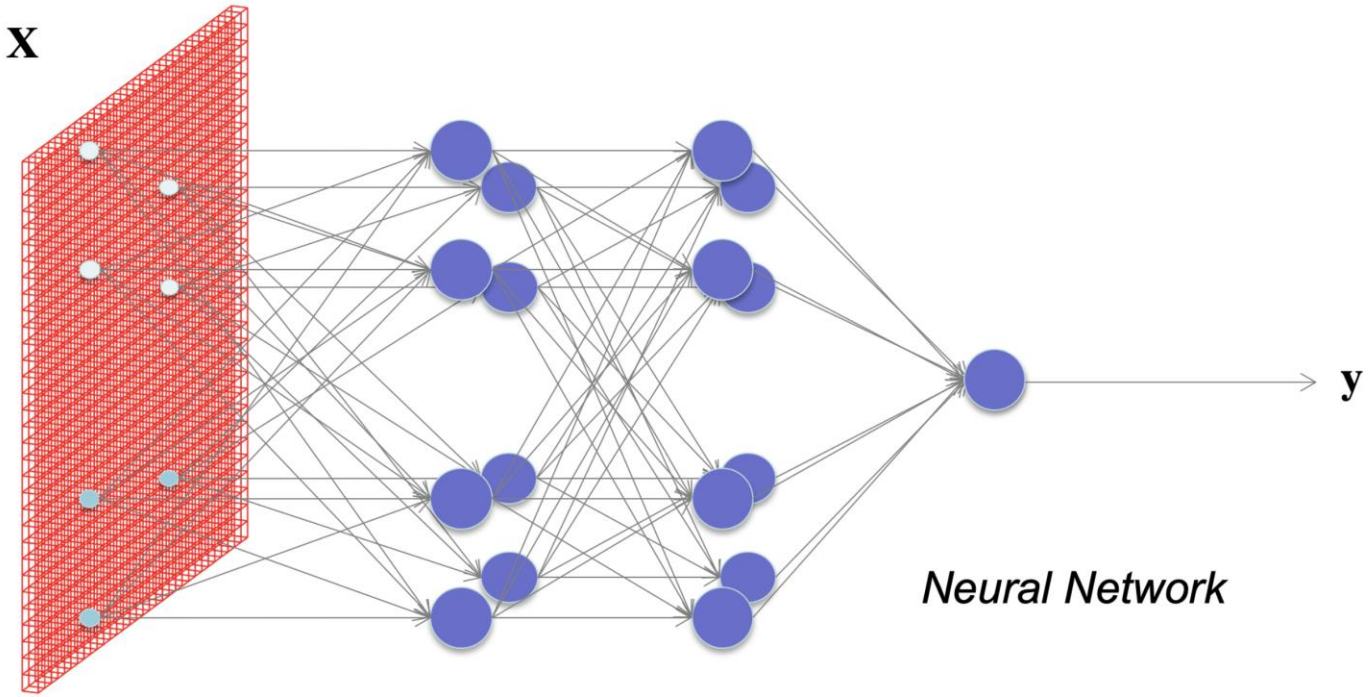
MOTIVATION

- How to process an image with neural networks?



MOTIVATION

- What happens if each pixel is an input?

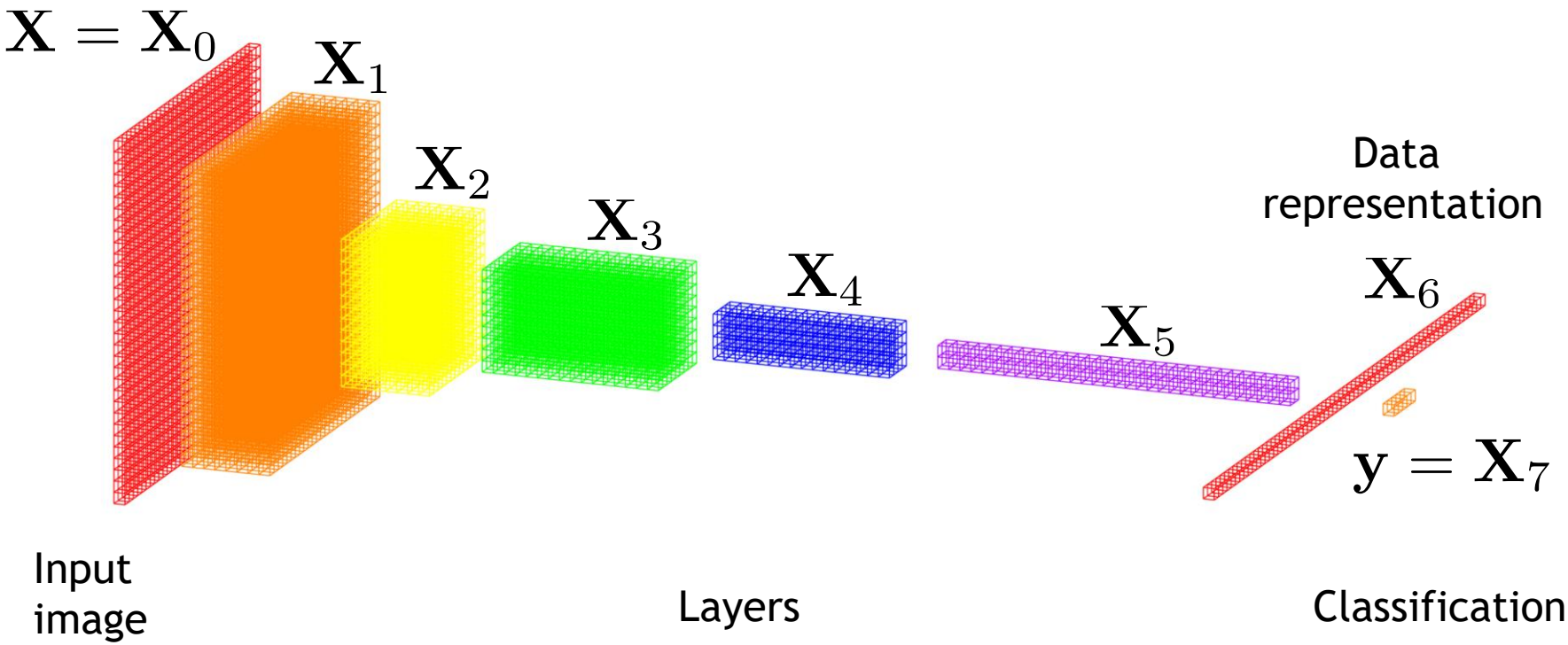


How many neurons and weights?

Basics

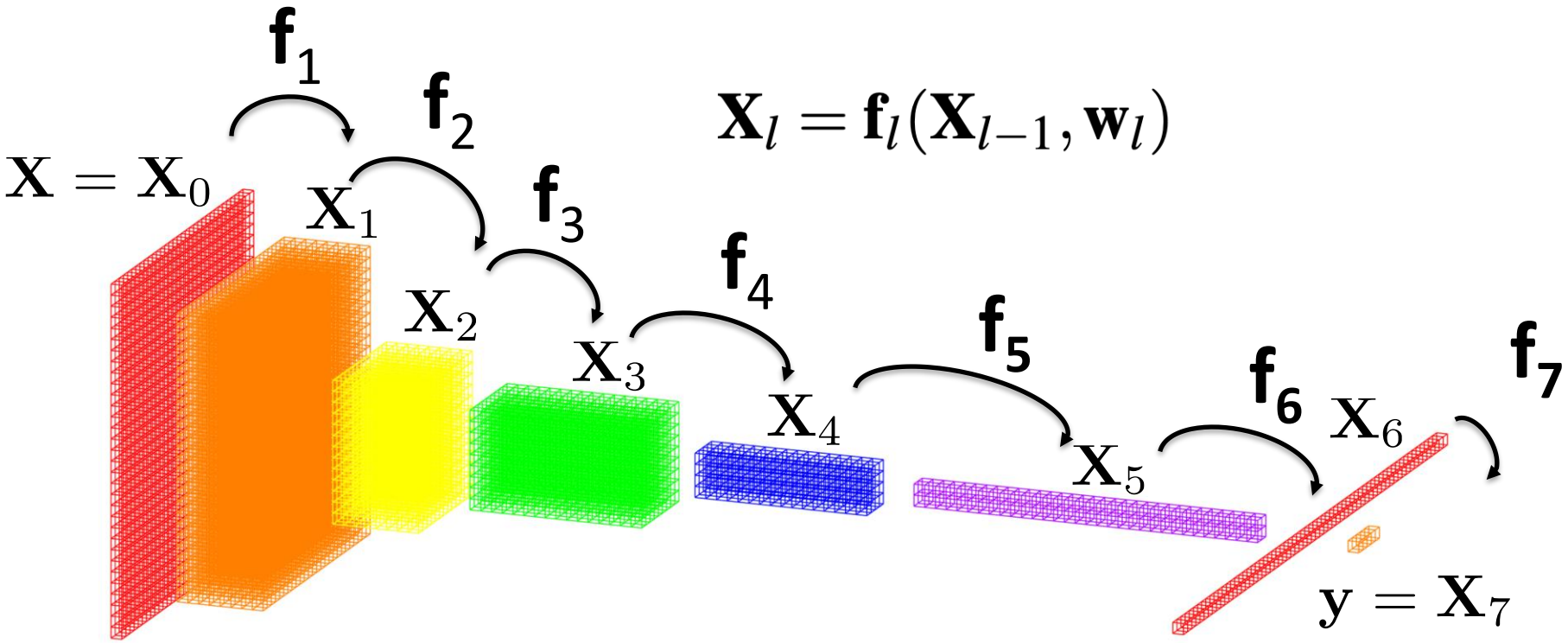
DEEP LEARNING BASICS (CNN)

Convolutional Neural Networks)



DEEP LEARNING BASICS (CNN)

Convolutional Neural Networks)



Input image

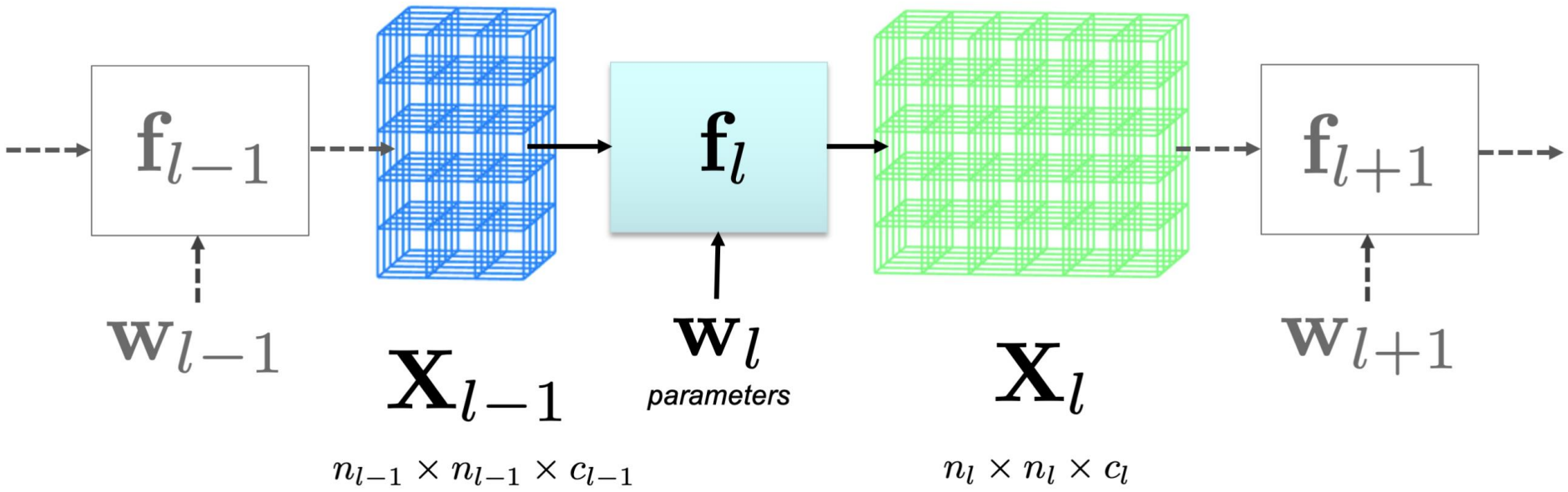
Layers

Classification

DEEP LEARNING BASICS

The convolutional neural network (CNN) can be understood as a set of layers, in which each layer processes an input image $\mathbf{X}$ in order to obtain an output image $\mathbf{Y}$:

$$\mathbf{X}_l = \mathbf{f}_l(\mathbf{X}_{l-1}, \mathbf{w}_l) \tag{1}$$



DEEP LEARNING: Typical layers

- Convolution layer
- Pooling layer
- Rectified Linear Unit
- Fully connected and SoftMax

DEEP LEARNING: Typical layers

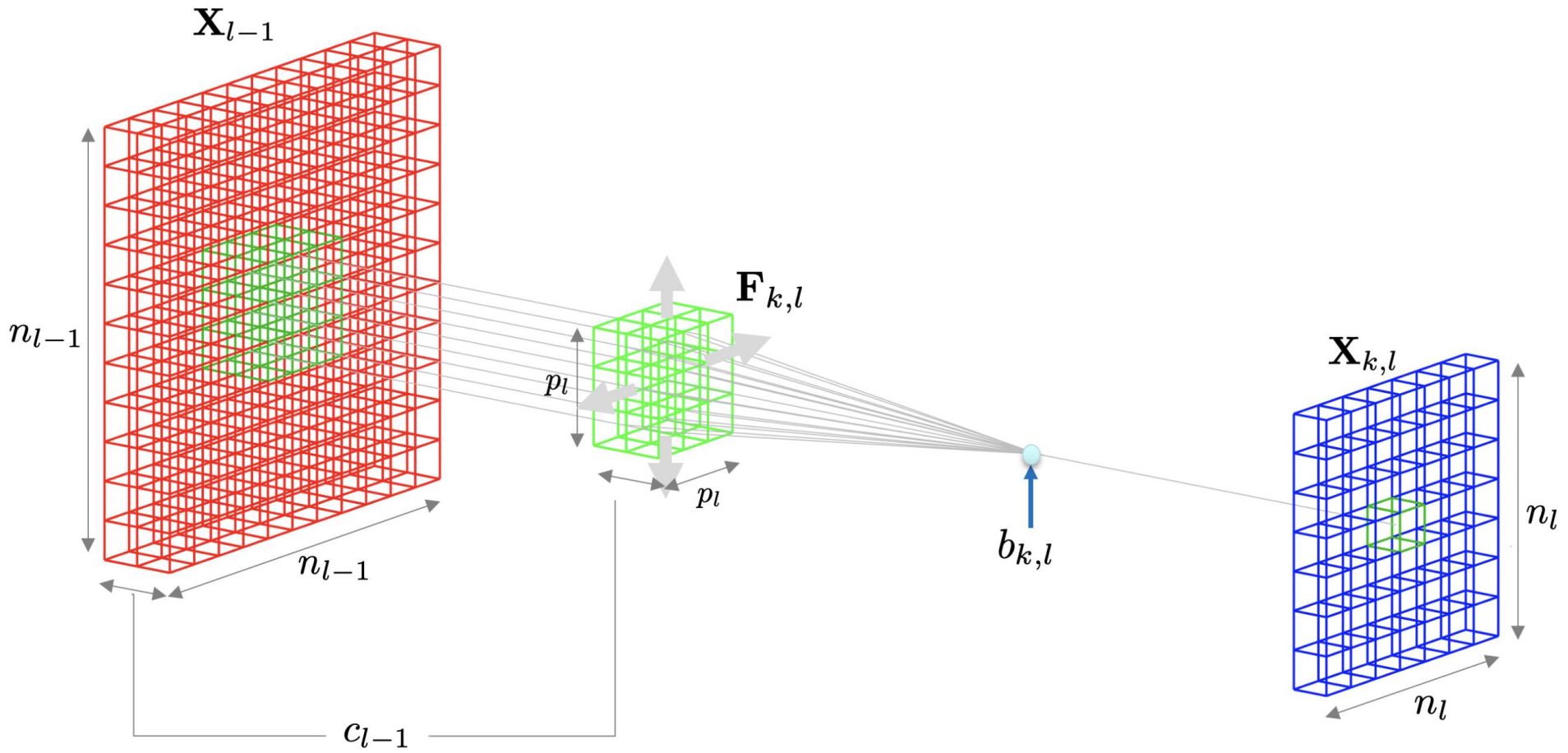
- Convolution layer
- Pooling layer
- Rectified Linear Unit
- Fully connected and SoftMax

DEEP LEARNING: Convolution layer

$$\mathbf{X}_{k,l} = \mathbf{X}_{l-1} * \mathbf{F}_{k,l} + b_{k,l} \quad \text{for } k = 1 \ldots m_l$$

$$X_{k,l}(i,j) = b_{k,l} + \sum_{u=1}^{p_l} \sum_{v=1}^{p_l} \sum_{w=1}^{q_l} X_{w,l-1}(i+u, j+v) F_{k,l}(u,v,w)$$

DEEP LEARNING: Convolution layer



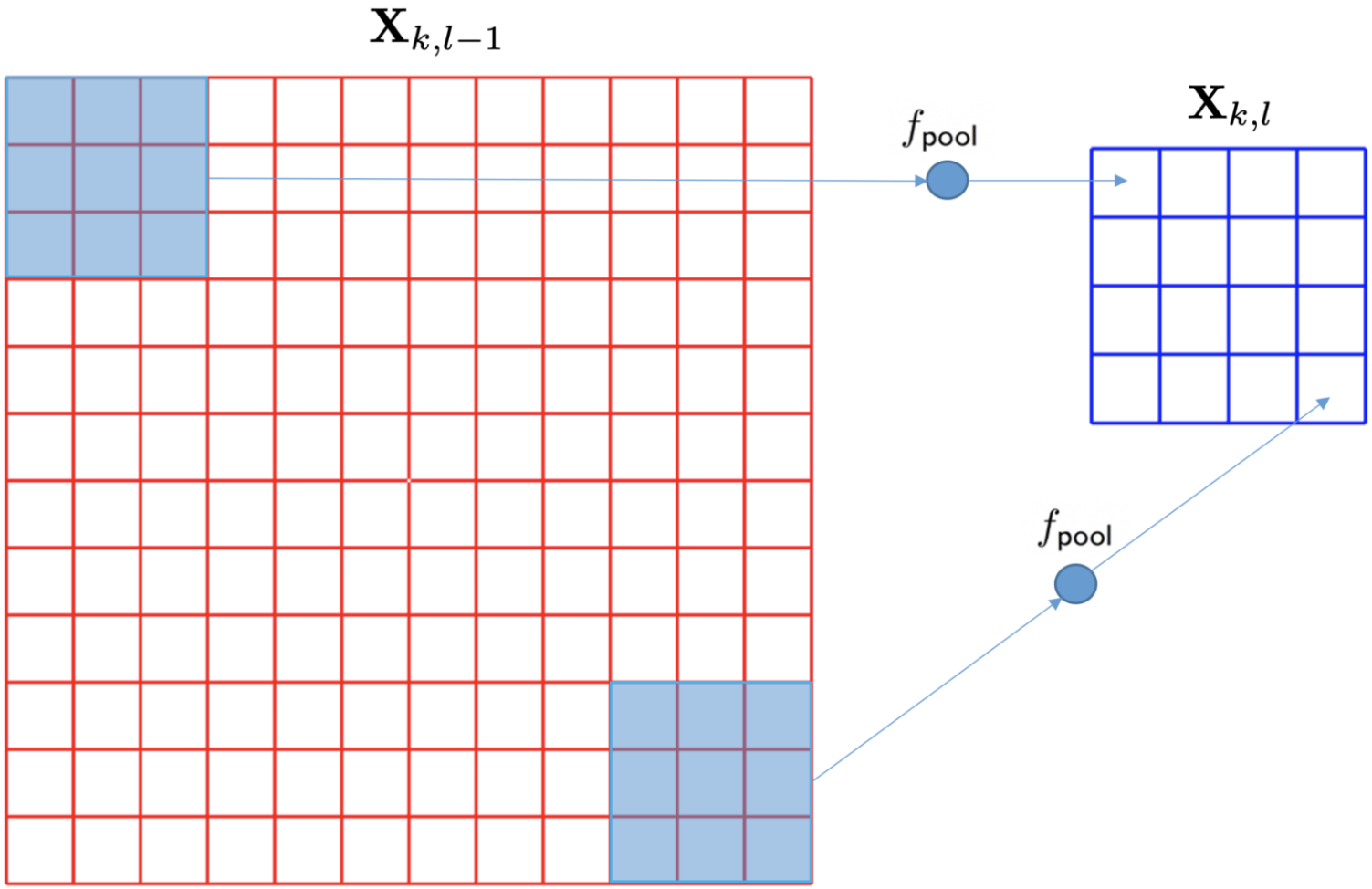
DEEP LEARNING: Typical layers

- Convolution layer
- Pooling layer
- Rectified Linear Unit
- Fully connected and SoftMax

DEEP LEARNING: Pooling layer

$$X_{k,l}(i,j) = f_{\text{pool}}\{X_{k,l-1}(u,v) : (u,v) \in \Omega(i,j)\}$$

DEEP LEARNING: Pooling layer



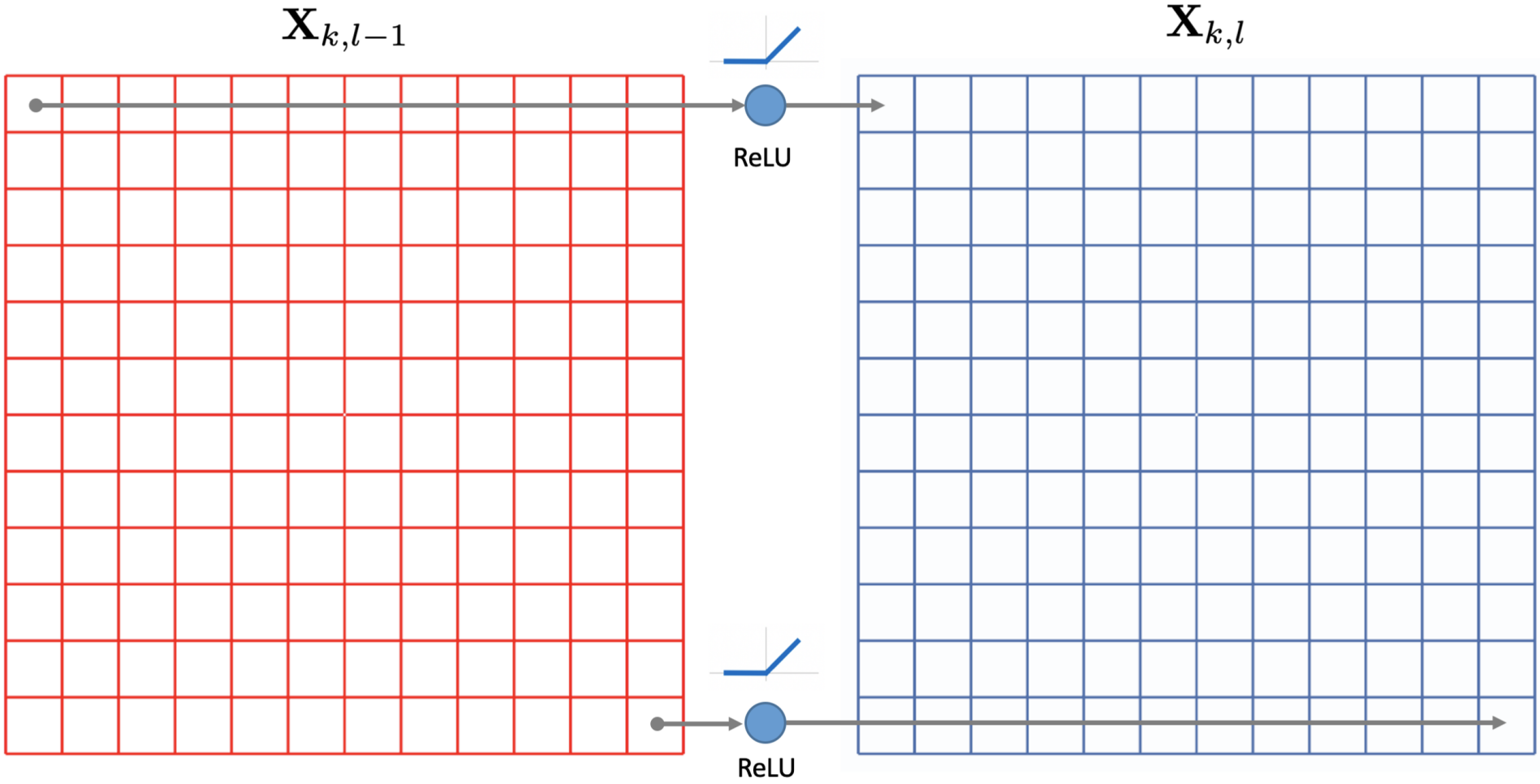
DEEP LEARNING: Typical layers

- Convolution layer
- Pooling layer
- Rectified Linear Unit
- Fully connected and SoftMax

DEEP LEARNING: Rectified Linear Unit (ReLU)

$$X_{k,l}(i,j) = \max\{0, X_{k,l-1}(i,j)\}$$

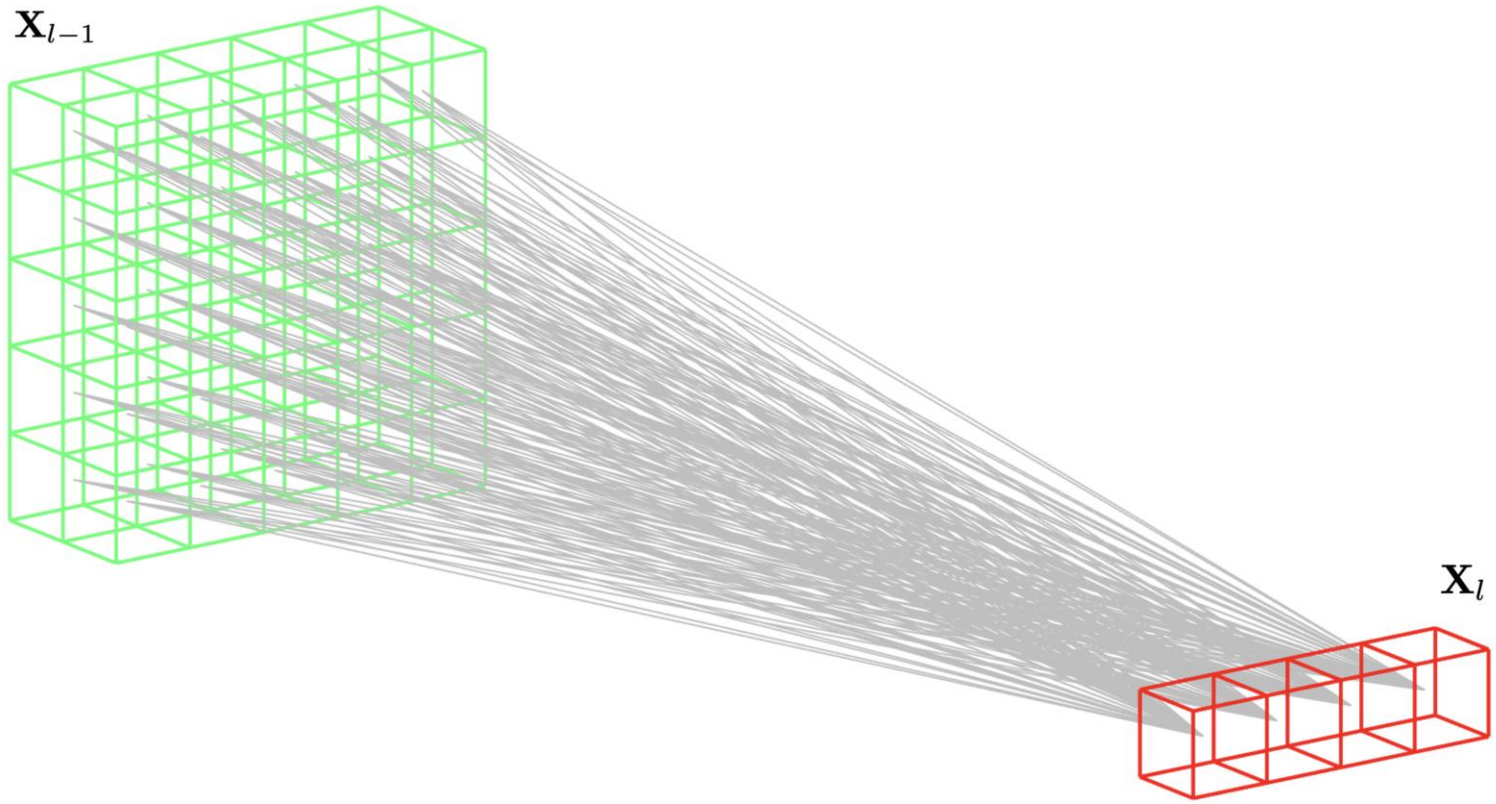
DEEP LEARNING: Rectified Linear Unit (ReLU)



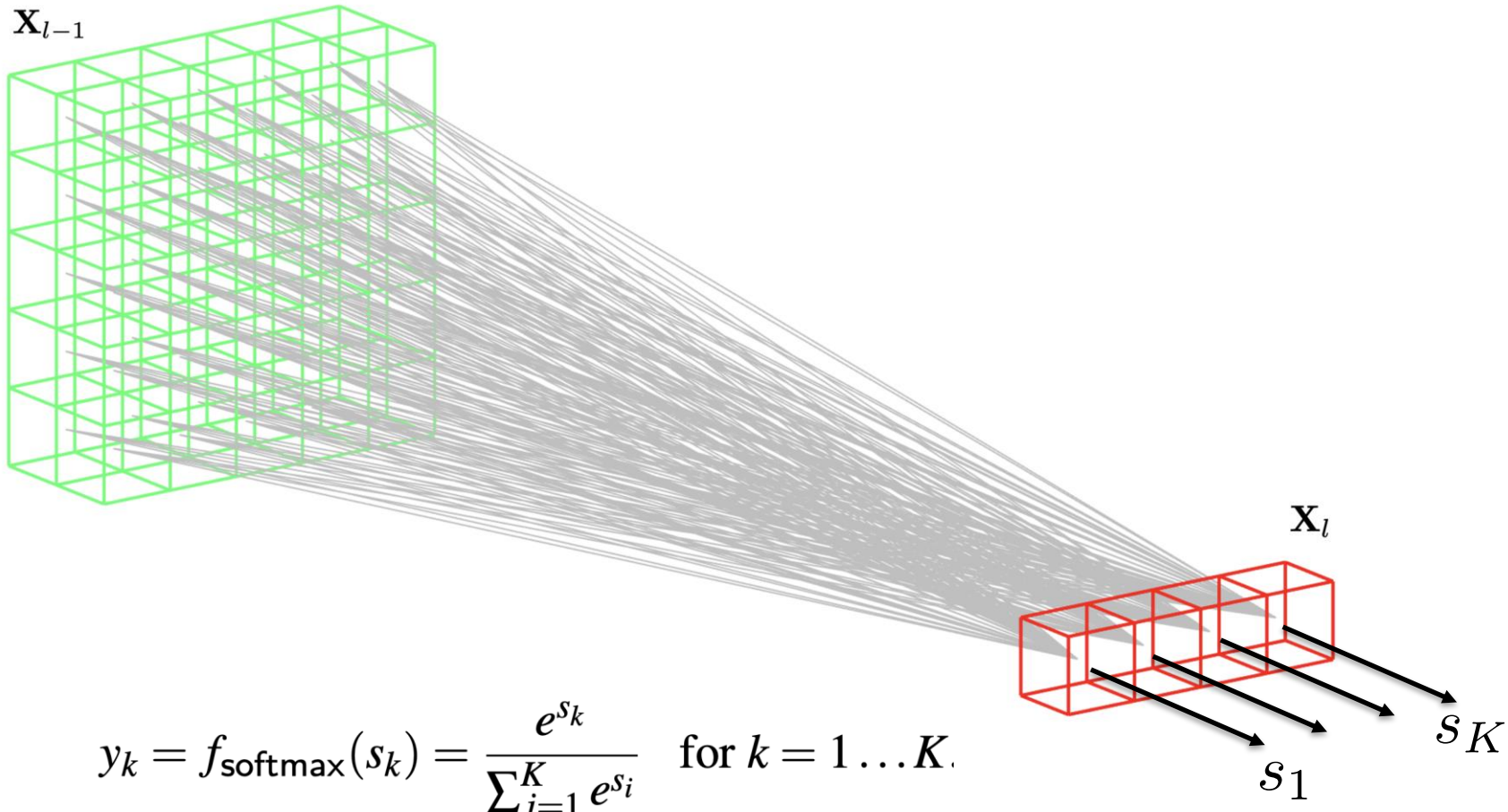
DEEP LEARNING: Typical layers

- Convolution layer
- Pooling layer
- Rectified Linear Unit
- Fully connected and SoftMax

DEEP LEARNING: Fully Connected



DEEP LEARNING: Fully Connected + SoftMax



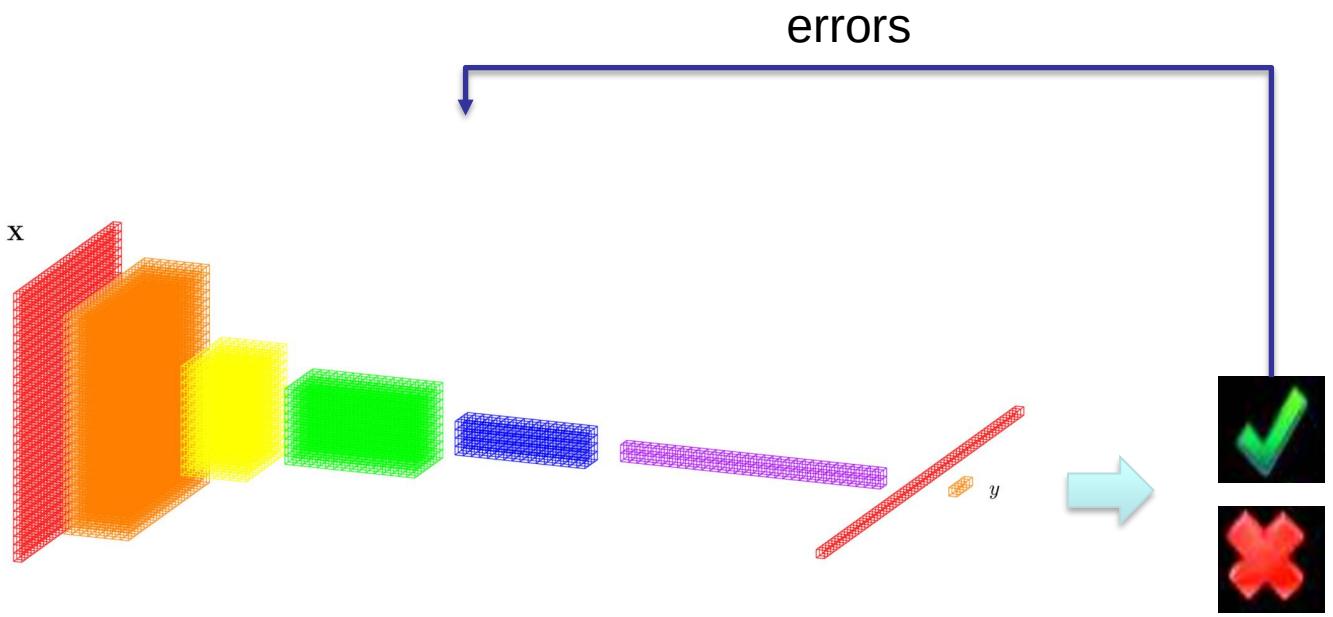
$$y_k = f_{\text{softmax}}(s_k) = \frac{e^{s_k}}{\sum_{j=1}^{K} e^{s_i}} \quad \text{for } k = 1 \ldots K.$$

Training

DEEP LEARNING: Training

dogs

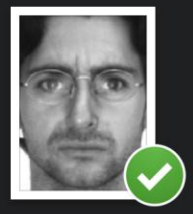
cats

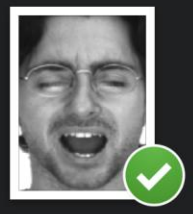


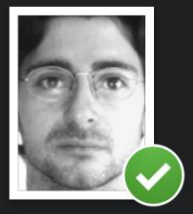
Back-propagation

Example

Eyes vs. Nose Recognizer


face_006_16.png


face_006_17.png


face_006_18.png

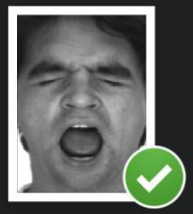

face_006_19.png

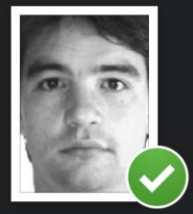

face_007_01.png

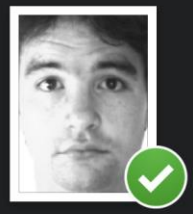

face_007_02.png

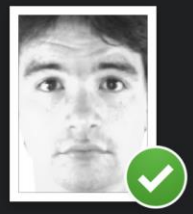

face_007_03.png

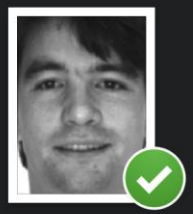

face_007_04.png

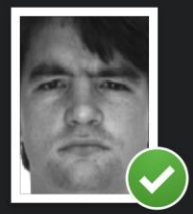

face_009_04.png

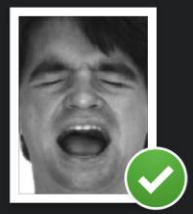

face_009_05.png

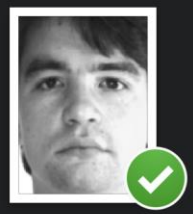

face_009_06.png


face_009_07.png

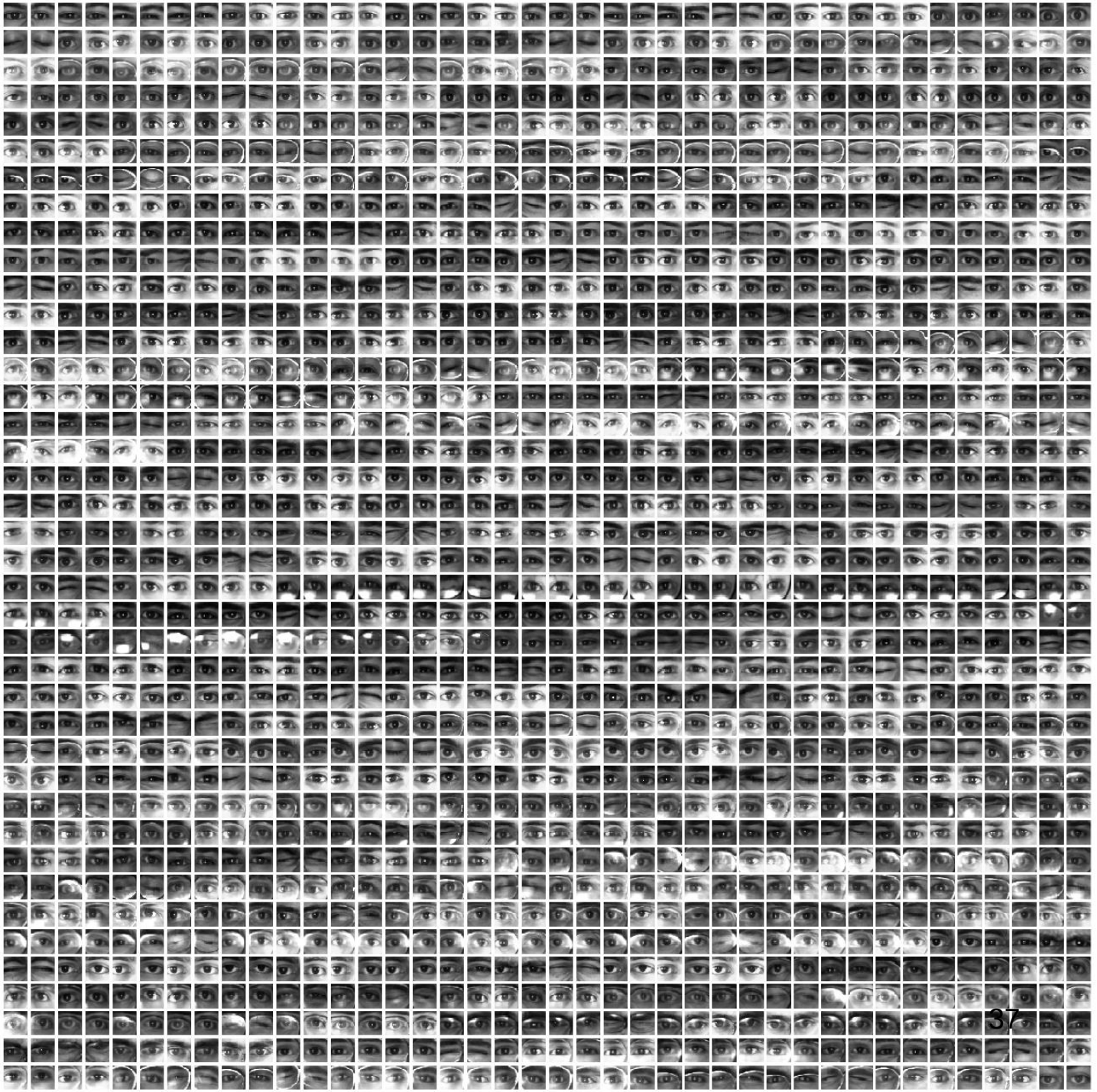

face_009_15.png


face_009_16.png

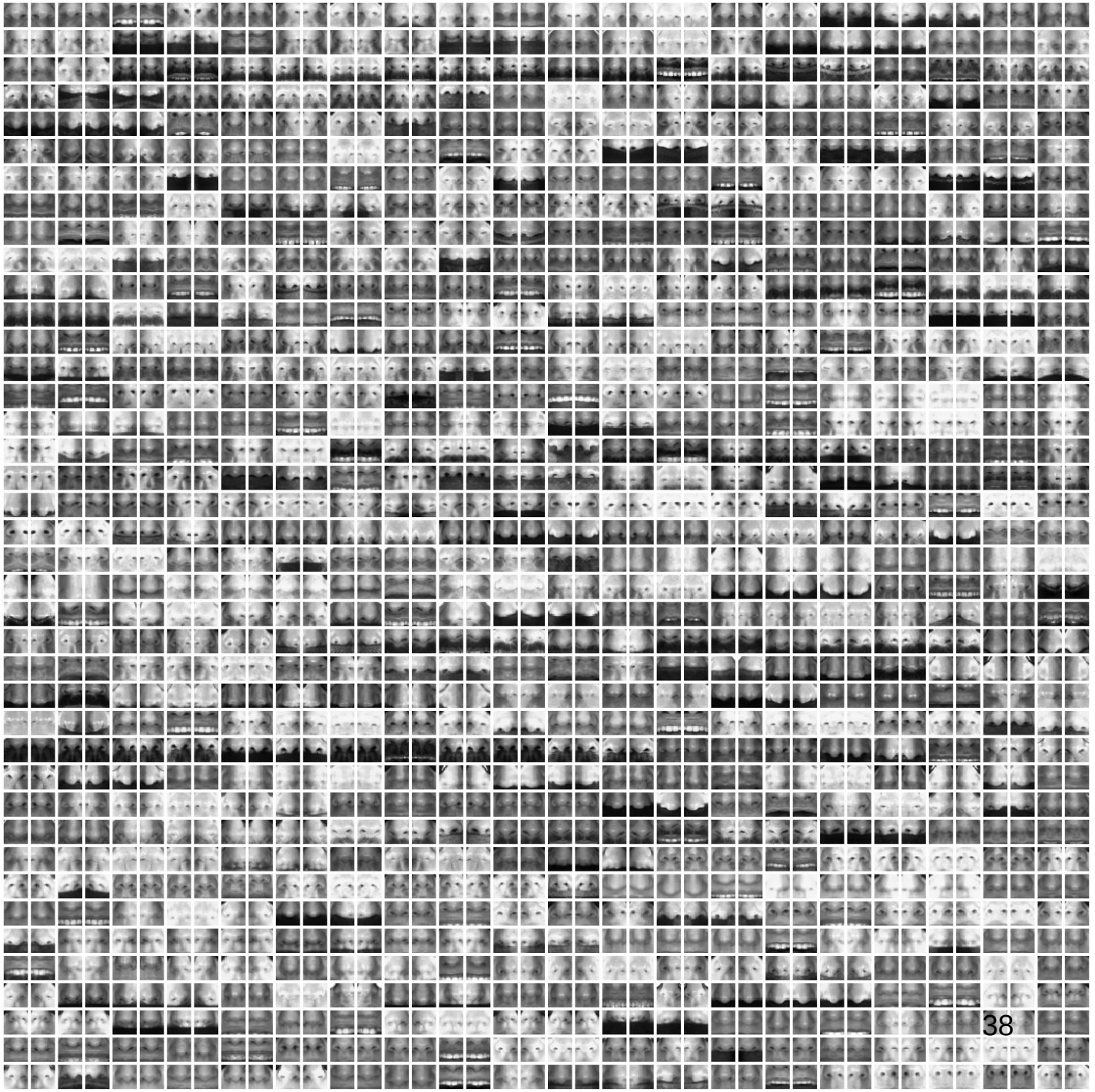

face_009_17.png

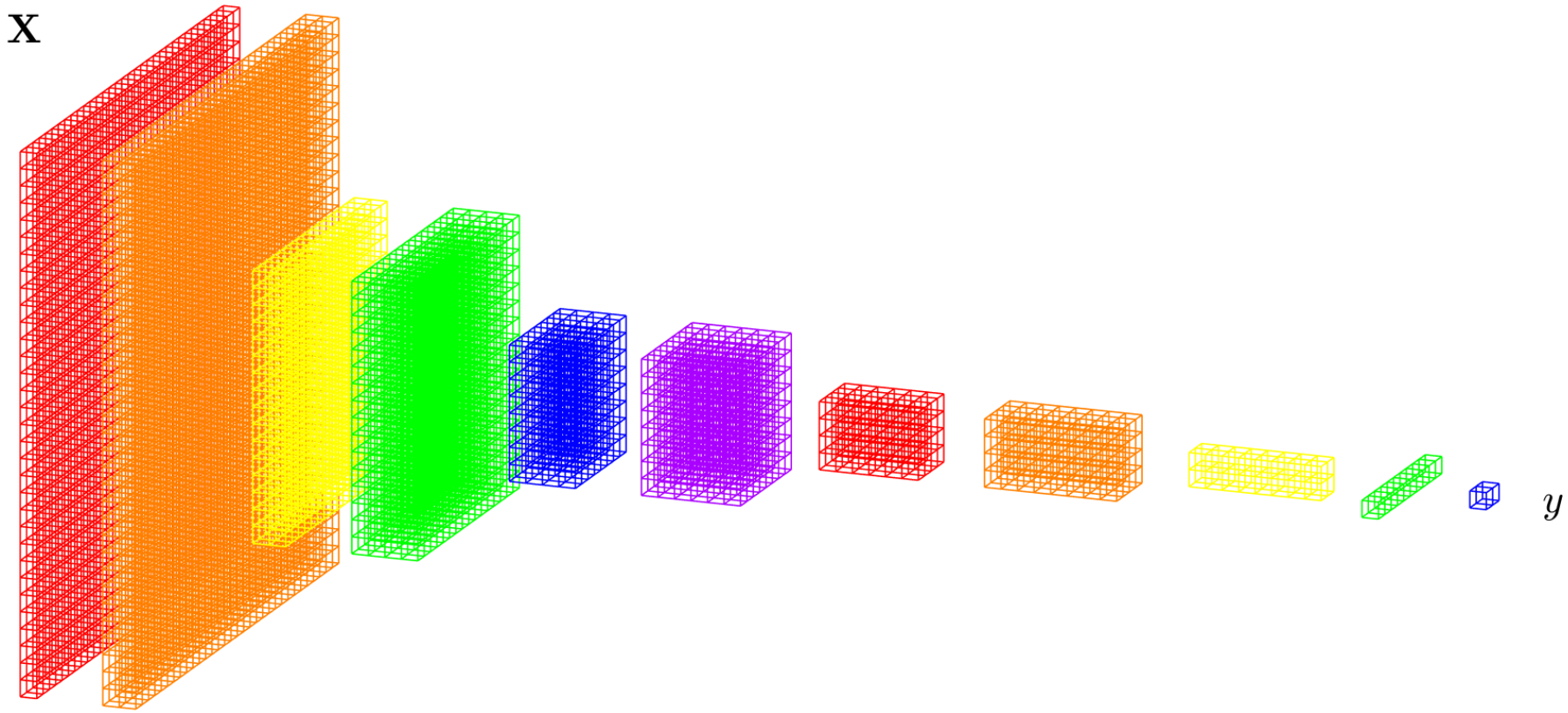

face_009_18.png

Eyes



Noses




$\mathbf{x}$
y

Implementation in PyTorch

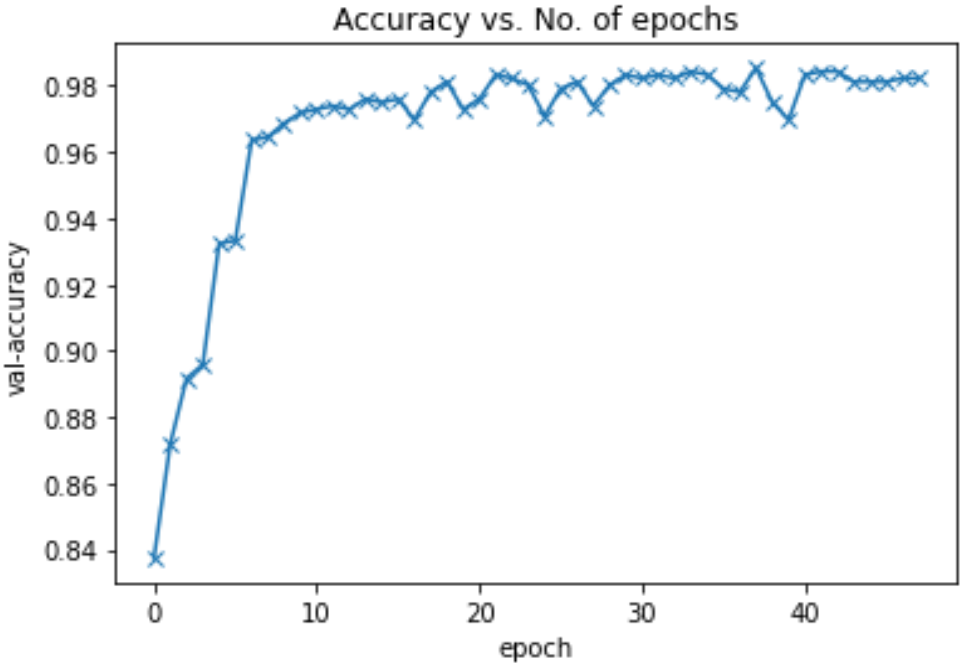
```
class CNN_Classification(ImageClassificationBase):
    def __init__(self):
        super().__init__()
        self.network = nn.Sequential(

            nn.Conv2d(3, 4, kernel_size = 5, stride = 1, padding = 0),
            nn.ReLU(),
            nn.Conv2d(4,8, kernel_size = 5, stride = 6, padding = 0),
            nn.ReLU(),
            nn.MaxPool2d(2,2),

            nn.Flatten(),
            nn.Linear(32,2)
        )

    def forward(self, xb):
        return self.network(xb)
```

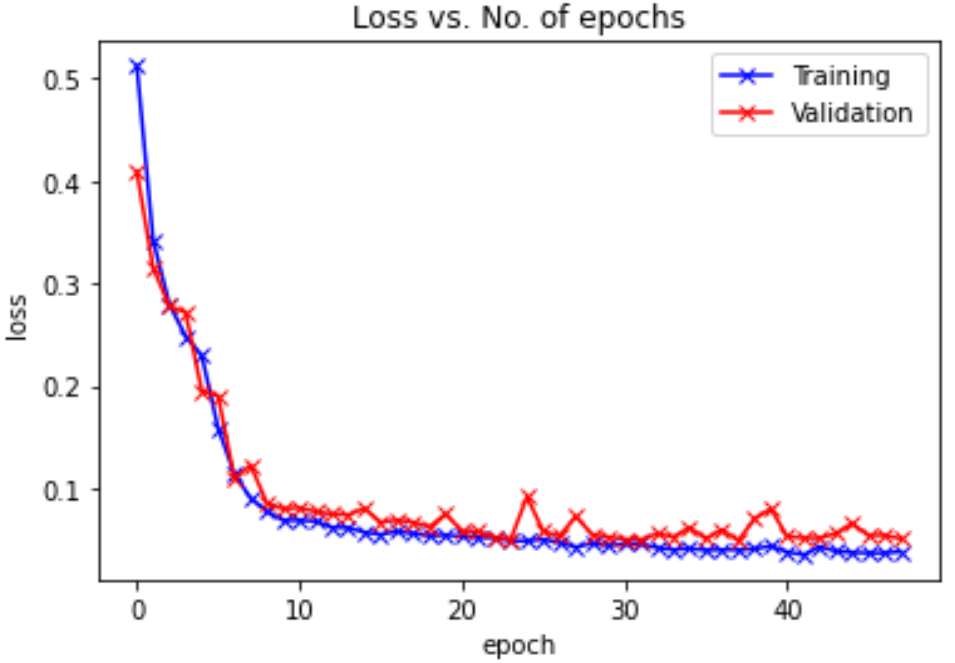
Epoch	Train-Loss	Val-Loss	Val-Acc	Best	Time [sec]
0	0.5126	0.4093	0.8375	***	1.8
1	0.3427	0.3156	0.8719	***	1.8
2	0.2797	0.2781	0.8917	***	1.8
3	0.2486	0.2719	0.8958	***	1.8
4	0.2295	0.1950	0.9323	***	1.8
5	0.1588	0.1904	0.9333	***	1.8
6	0.1145	0.1096	0.9635	***	1.8
7	0.0904	0.1229	0.9646	***	1.8
8	0.0776	0.0855	0.9687	***	1.8
9	0.0688	0.0801	0.9719	***	1.8
10	0.0689	0.0811	0.9729	***	1.8
11	0.0681	0.0774	0.9740	***	1.8
12	0.0620	0.0750	0.9729		1.8
13	0.0617	0.0742	0.9760	***	1.8
14	0.0572	0.0806	0.9750		1.8
15	0.0551	0.0670	0.9760		1.8
16	0.0580	0.0692	0.9698		1.8
17	0.0560	0.0671	0.9781	***	1.8
18	0.0538	0.0624	0.9813	***	1.8
19	0.0544	0.0752	0.9729		1.8
20	0.0533	0.0582	0.9760		1.8
21	0.0522	0.0578	0.9833	***	1.9
22	0.0519	0.0526	0.9823		1.8
23	0.0489	0.0503	0.9802		1.8
24	0.0488	0.0918	0.9708		1.8
25	0.0504	0.0582	0.9792		1.8
26	0.0470	0.0535	0.9813		1.8
27	0.0428	0.0728	0.9740		1.8
28	0.0463	0.0551	0.9802		1.8
29	0.0443	0.0527	0.9833	***	1.8
30	0.0459	0.0504	0.9823		1.8
31	0.0461	0.0501	0.9833		1.8
32	0.0420	0.0557	0.9823		1.8
33	0.0402	0.0528	0.9844	***	1.8
34	0.0414	0.0619	0.9833		1.8
35	0.0397	0.0510	0.9792		1.9
36	0.0403	0.0597	0.9781		1.8
37	0.0400	0.0491	0.9854	***	1.8
38	0.0410	0.0704	0.9750		1.8
39	0.0438	0.0802	0.9698		1.8
40	0.0382	0.0532	0.9833		1.9
41	0.0350	0.0517	0.9844		1.8
42	0.0432	0.0516	0.9844		1.8
43	0.0387	0.0561	0.9813		1.8
44	0.0374	0.0660	0.9813		1.8
45	0.0370	0.0540	0.9813		1.8
46	0.0369	0.0544	0.9823		1.8
47	0.0383	0.0509	0.9823		1.8

*** Early stop after 10 epochs with no improvement

Best model saved best_model.pt (val_acc = 0.9854 in epoch = 37)
Last model saved last_model.pt (val_acc = 0.9823 in epoch = 47)

Training Time: 85.82 sec





Epoch	Train-Loss	Val-Loss	Val-Acc	Best	Time [sec]
0	0.5126	0.4093	0.8375	***	1.8
1	0.3427	0.3156	0.8719	***	1.8
2	0.2797	0.2781	0.8917	***	1.8
3	0.2486	0.2719	0.8958	***	1.8
4	0.2295	0.1950	0.9323	***	1.8
5	0.1588	0.1904	0.9333	***	1.8
6	0.1145	0.1096	0.9635	***	1.8
7	0.0904	0.1229	0.9646	***	1.8
8	0.0776	0.0855	0.9687	***	1.8
9	0.0688	0.0801	0.9719	***	1.8
10	0.0689	0.0811	0.9729	***	1.8
11	0.0681	0.0774	0.9740	***	1.8
12	0.0620	0.0750	0.9729		1.8
13	0.0617	0.0742	0.9760	***	1.8
14	0.0572	0.0806	0.9750		1.8
15	0.0551	0.0670	0.9760		1.8
16	0.0580	0.0692	0.9698		1.8
17	0.0560	0.0671	0.9781	***	1.8
18	0.0538	0.0624	0.9813	***	1.8
19	0.0544	0.0752	0.9729		1.8
20	0.0533	0.0582	0.9760		1.8
21	0.0522	0.0578	0.9833	***	1.9
22	0.0519	0.0526	0.9823		1.8
23	0.0489	0.0503	0.9802		1.8
24	0.0488	0.0918	0.9708		1.8
25	0.0504	0.0582	0.9792		1.8
26	0.0470	0.0535	0.9813		1.8
27	0.0428	0.0728	0.9740		1.8
28	0.0463	0.0551	0.9802		1.8
29	0.0443	0.0527	0.9833	***	1.8
30	0.0459	0.0504	0.9823		1.8
31	0.0461	0.0501	0.9833		1.8
32	0.0420	0.0557	0.9823		1.8
33	0.0402	0.0528	0.9844	***	1.8
34	0.0414	0.0619	0.9833		1.8
35	0.0397	0.0510	0.9792		1.9
36	0.0403	0.0597	0.9781		1.8
37	0.0400	0.0491	0.9854	***	1.8
38	0.0410	0.0704	0.9750		1.8
39	0.0438	0.0802	0.9698		1.8
40	0.0382	0.0532	0.9833		1.9
41	0.0350	0.0517	0.9844		1.8
42	0.0432	0.0516	0.9844		1.8
43	0.0387	0.0561	0.9813		1.8
44	0.0374	0.0660	0.9813		1.8
45	0.0370	0.0540	0.9813		1.8
46	0.0369	0.0544	0.9823		1.8
47	0.0383	0.0509	0.9823		1.8

*** Early stop after 10 epochs with no improvement

Best model saved best_model.pt (val_acc = 0.9854 in epoch = 37)
Last model saved last_model.pt (val_acc = 0.9823 in epoch = 47)

Training Time: 85.82 sec

Training Confusion Matrix =
[[3616 45]
[36 3663]]

Training Accuracy = 0.9890

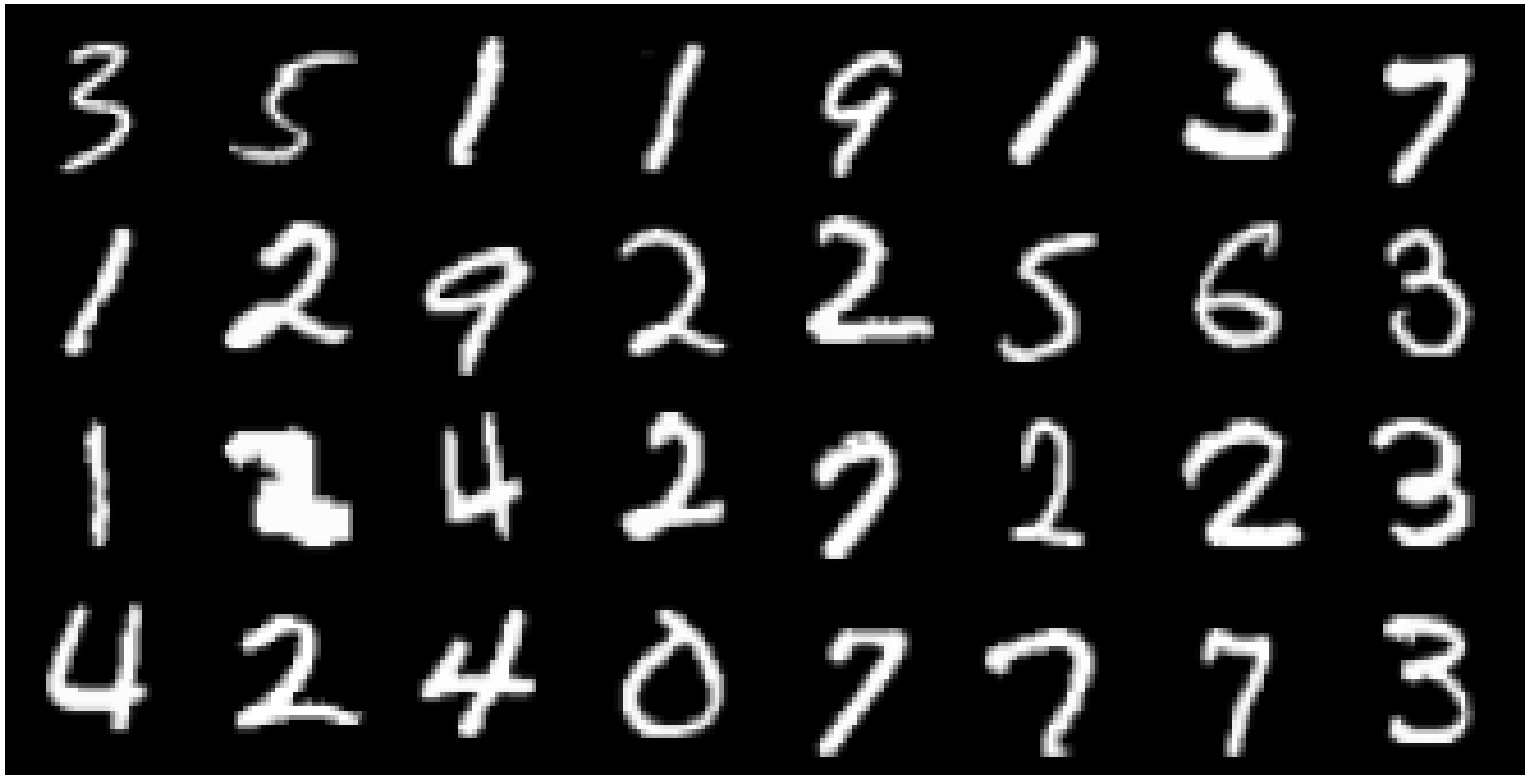
Validation Confusion Matrix =
[[490 9]
[5 456]]

Validation Accuracy = 0.9854

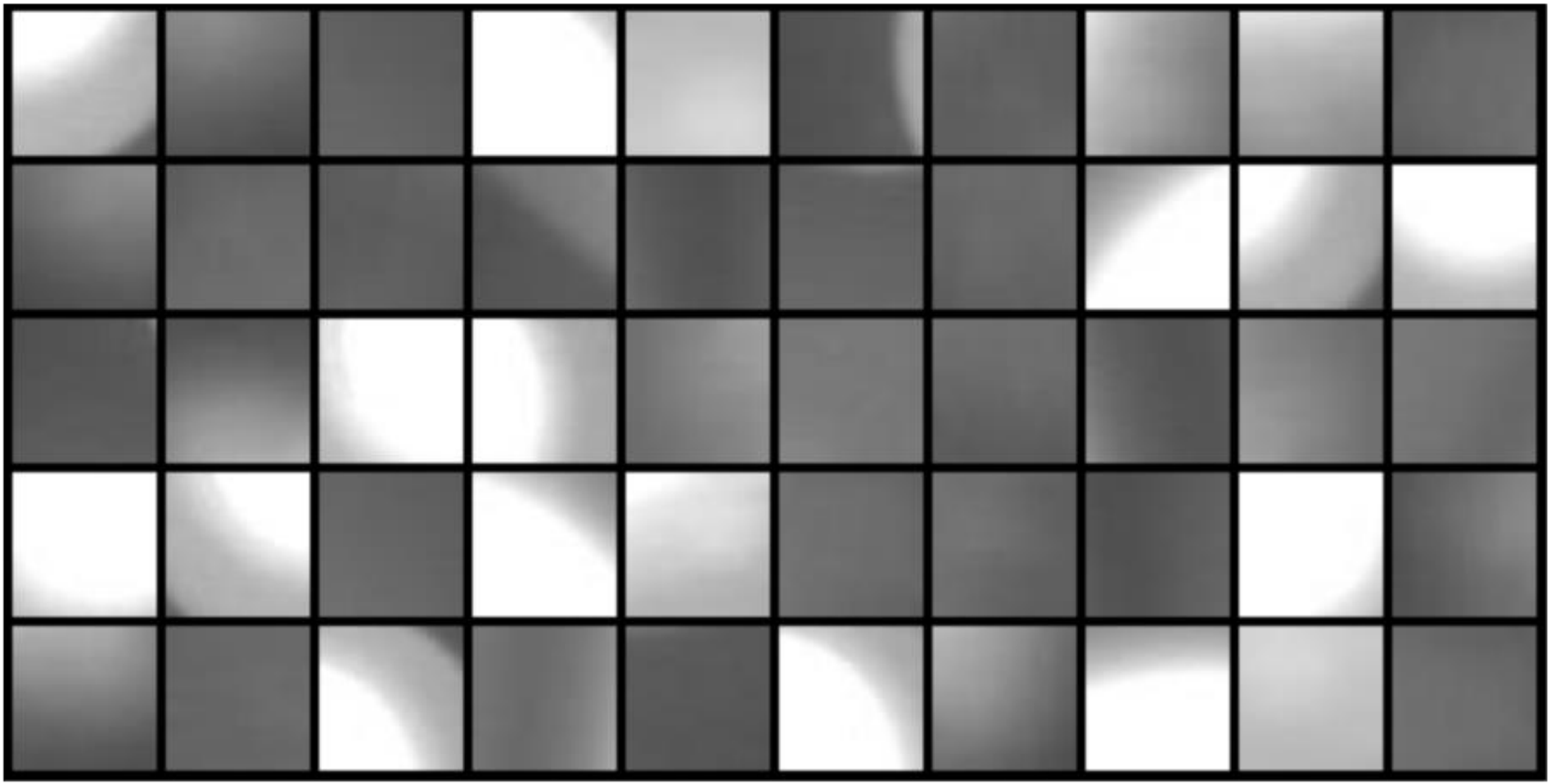
Testing Confusion Matrix =
[[1027 13]
[6 1034]]

Testing Accuracy = 0.9909

More Examples: MNIST

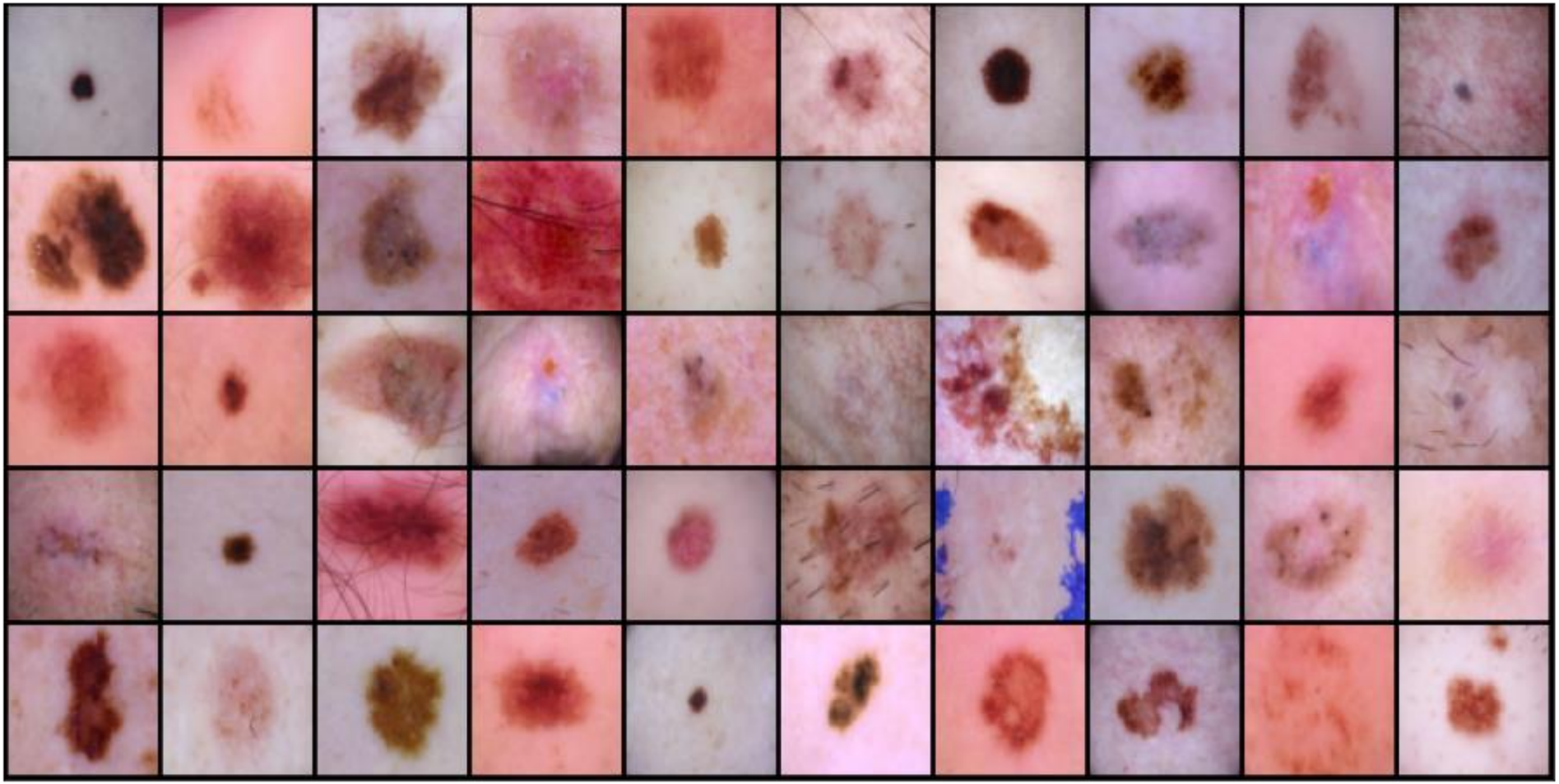


More Examples: Defects



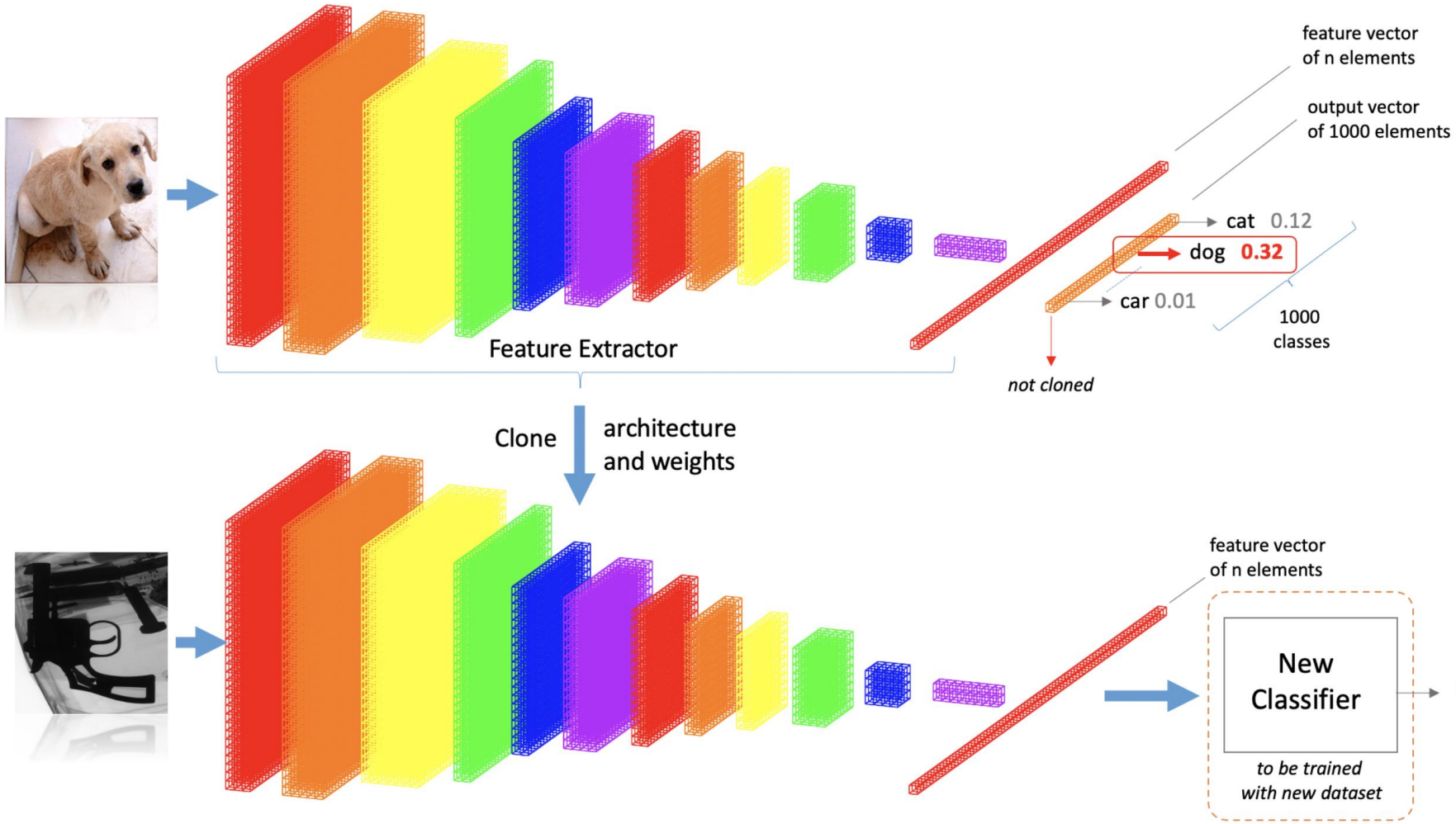
More Examples: Cats vs Dogs

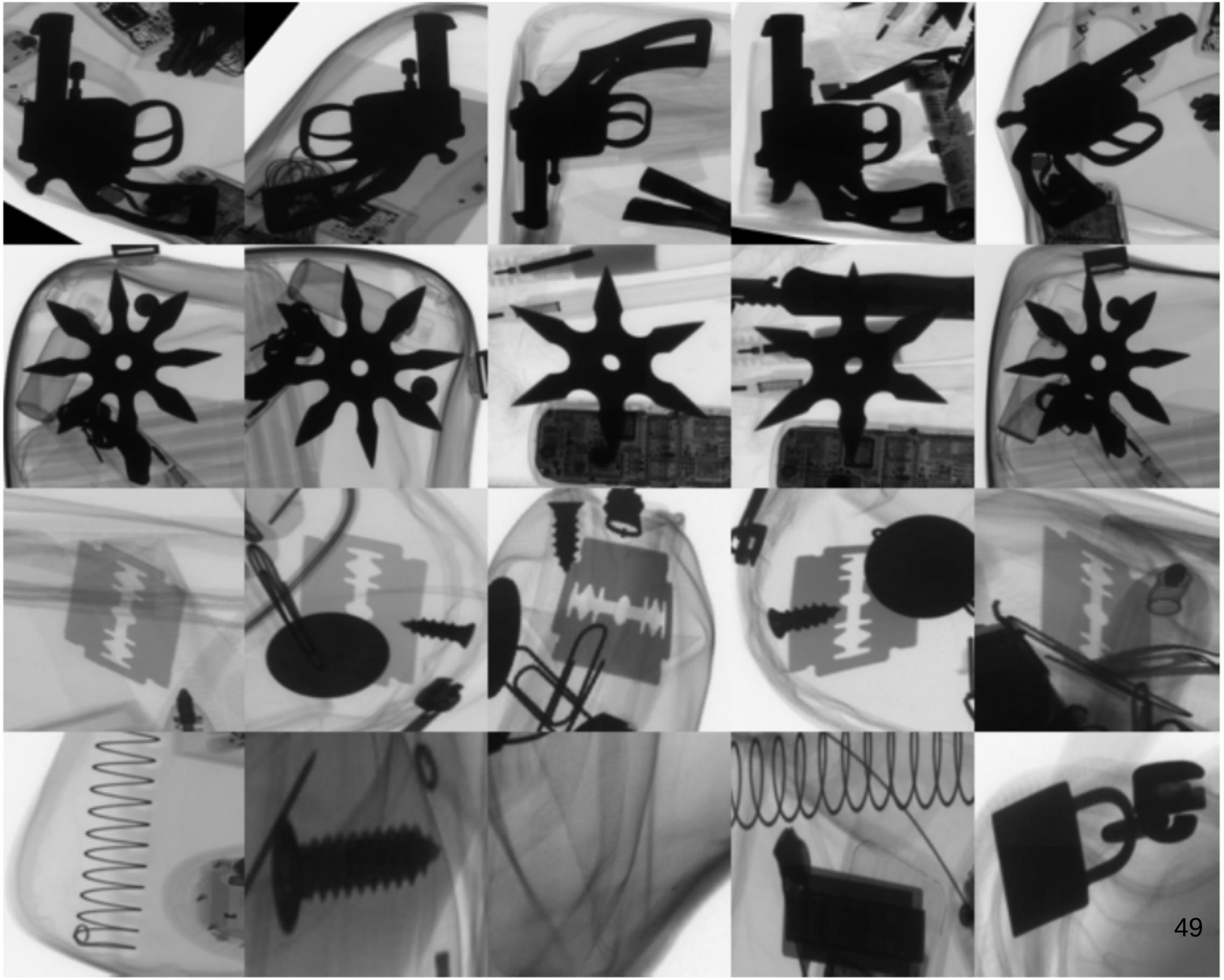
More Examples: Skin Cancer 2



More Examples: Skin Cancer 7

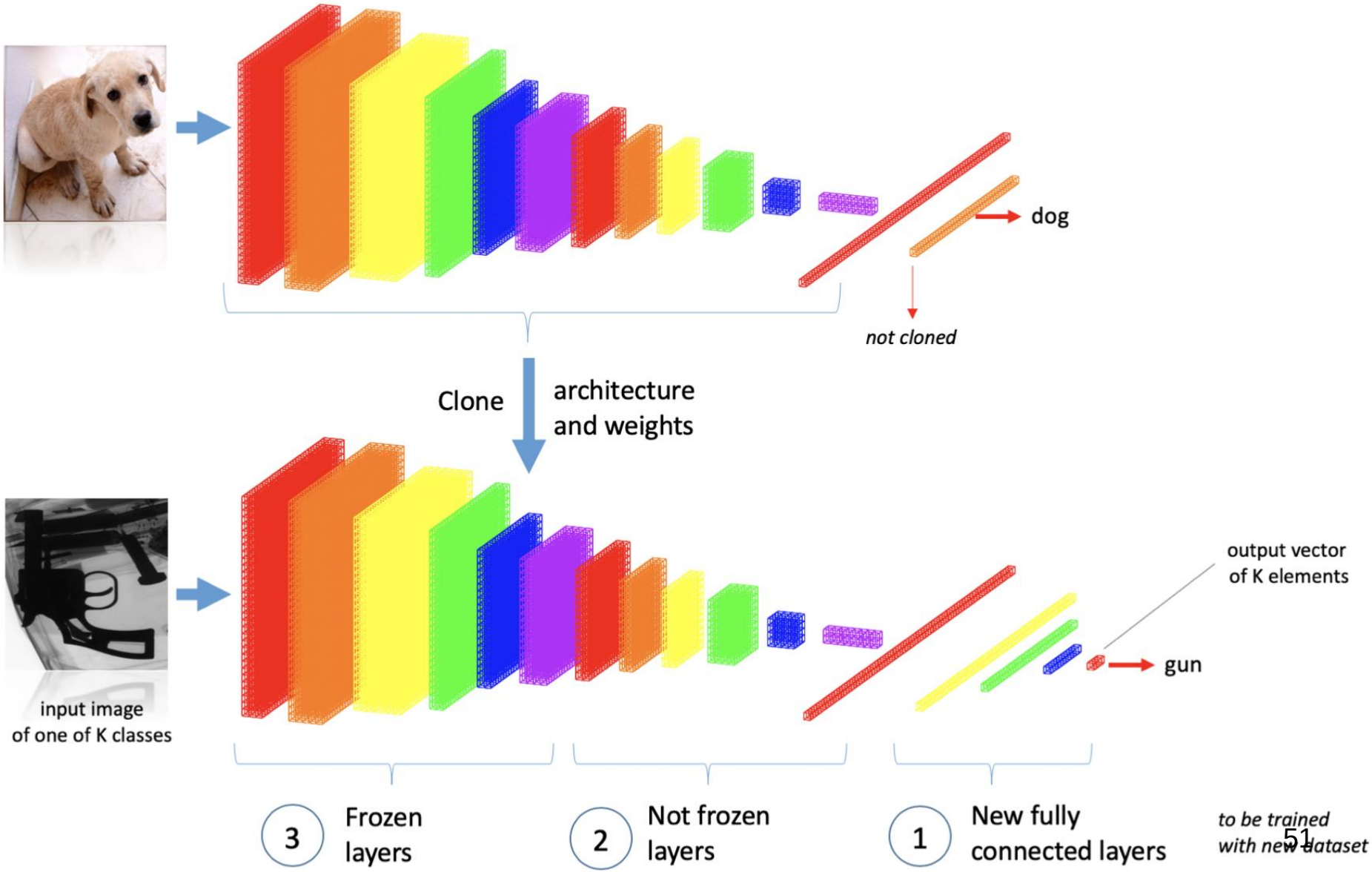
Pre-Trained Models



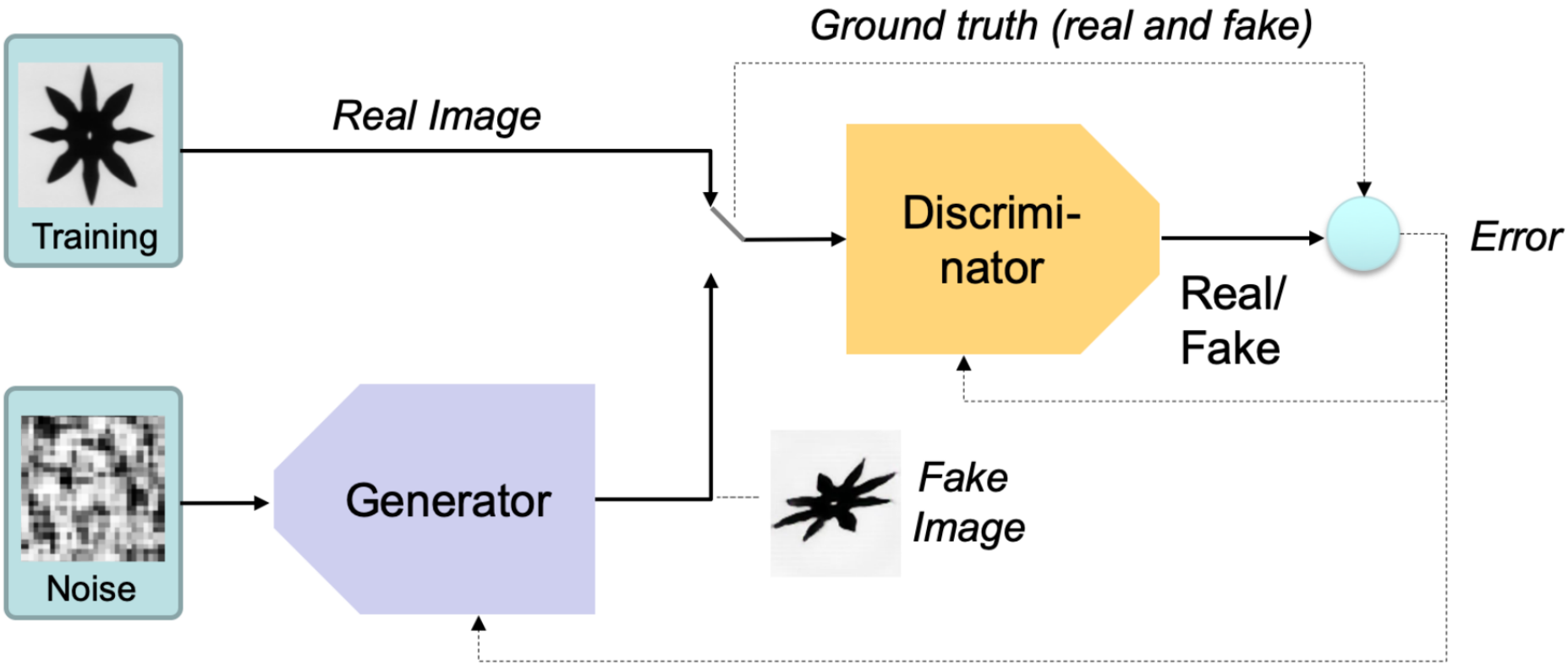


Method	Implementation	Features	Gun		Shuriken		Blade		Total
			Pr	Re	Pr	Re	Pr	Re	η
ResNet50	2	2048	1.00	0.90	1.00	1.00	0.35	0.55	0.80
VGG16	1	1000	0.95	0.86	0.56	1.00	0.36	0.84	0.73
VGG19	2	4096	0.78	0.72	0.27	1.00	0.18	0.58	0.53
AlexNet	2	4096	0.80	0.96	0.84	1.00	0.62	0.54	0.79
DenseNet121	1	1000	0.99	0.82	0.99	0.98	0.34	0.61	0.78
GoogleNet	2	1024	0.89	0.98	0.74	1.00	0.23	0.20	0.67
InceptionResNetV2	0	1536	0.90	0.60	0.82	0.99	0.55	0.60	0.74
InceptionV3	0	2048	0.85	0.69	0.76	1.00	0.47	0.40	0.69
MobileNet	2	1280	0.98	0.84	0.94	1.00	0.35	0.95	0.82
RCNN-ILSVRC13	2	4096	0.75	1.00	0.69	1.00	0.37	0.15	0.64
ShuffleNet	2	544	0.97	0.78	0.86	1.00	0.20	0.95	0.74
SqueezeNet	2	1000	0.96	0.38	0.27	1.00	0.06	0.35	0.42
Xception	0	2048	0.54	0.12	0.75	0.78	0.06	0.21	0.38
ZfNet512	2	1024	0.87	0.84	0.28	1.00	0.04	0.10	0.48

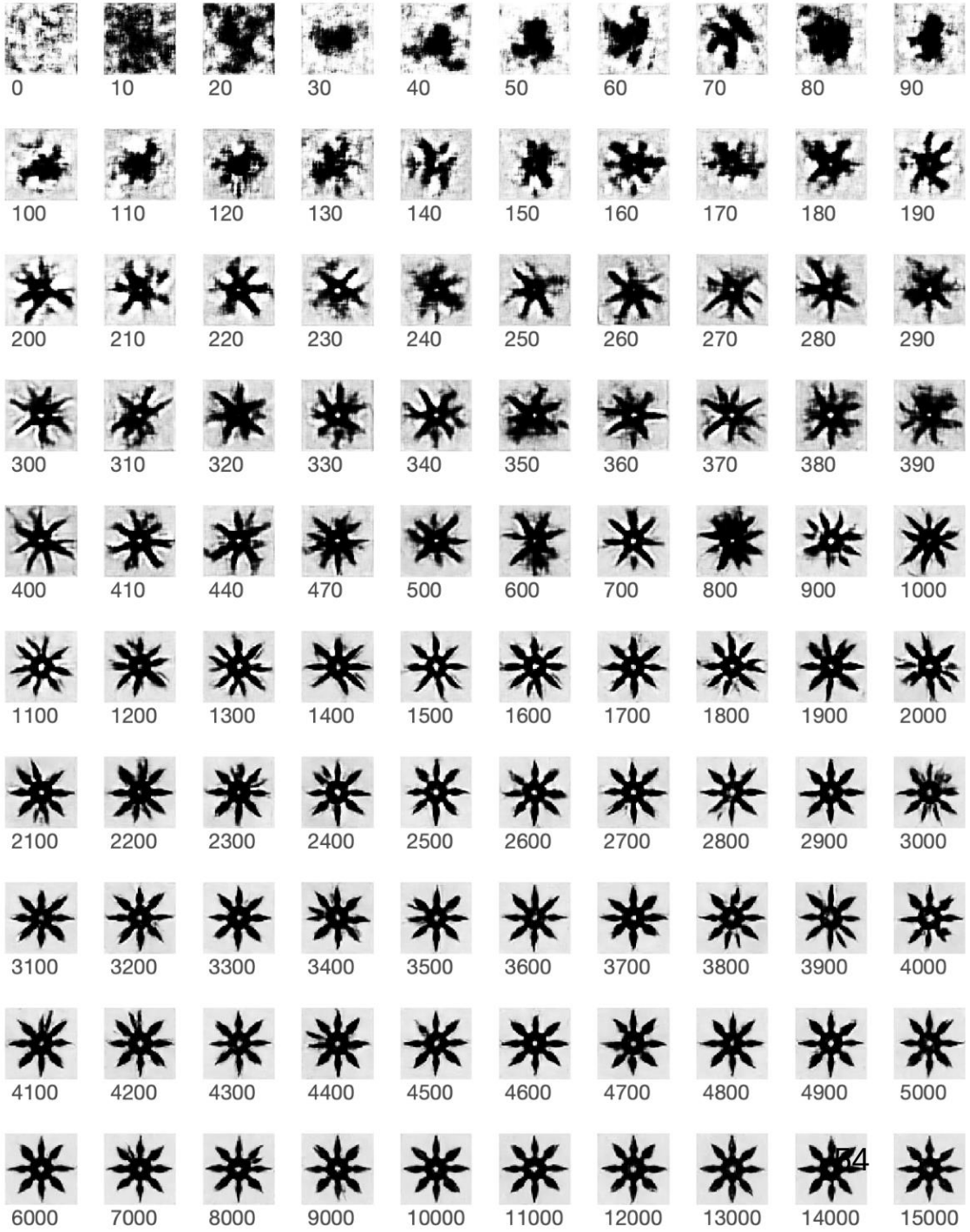
Transfer Learning



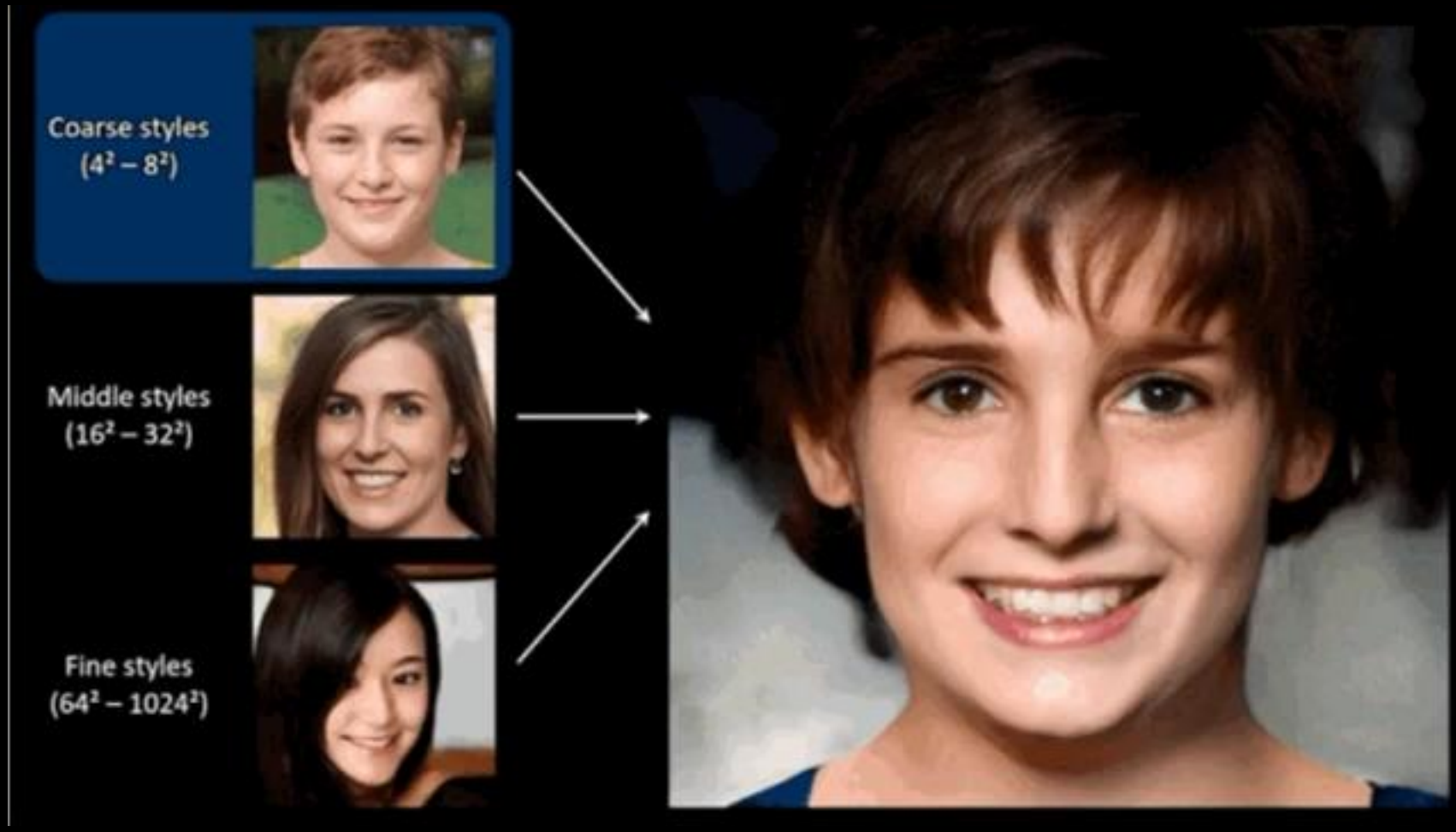
GAN: Generative Adversarial Networks



GAN: Shuriken

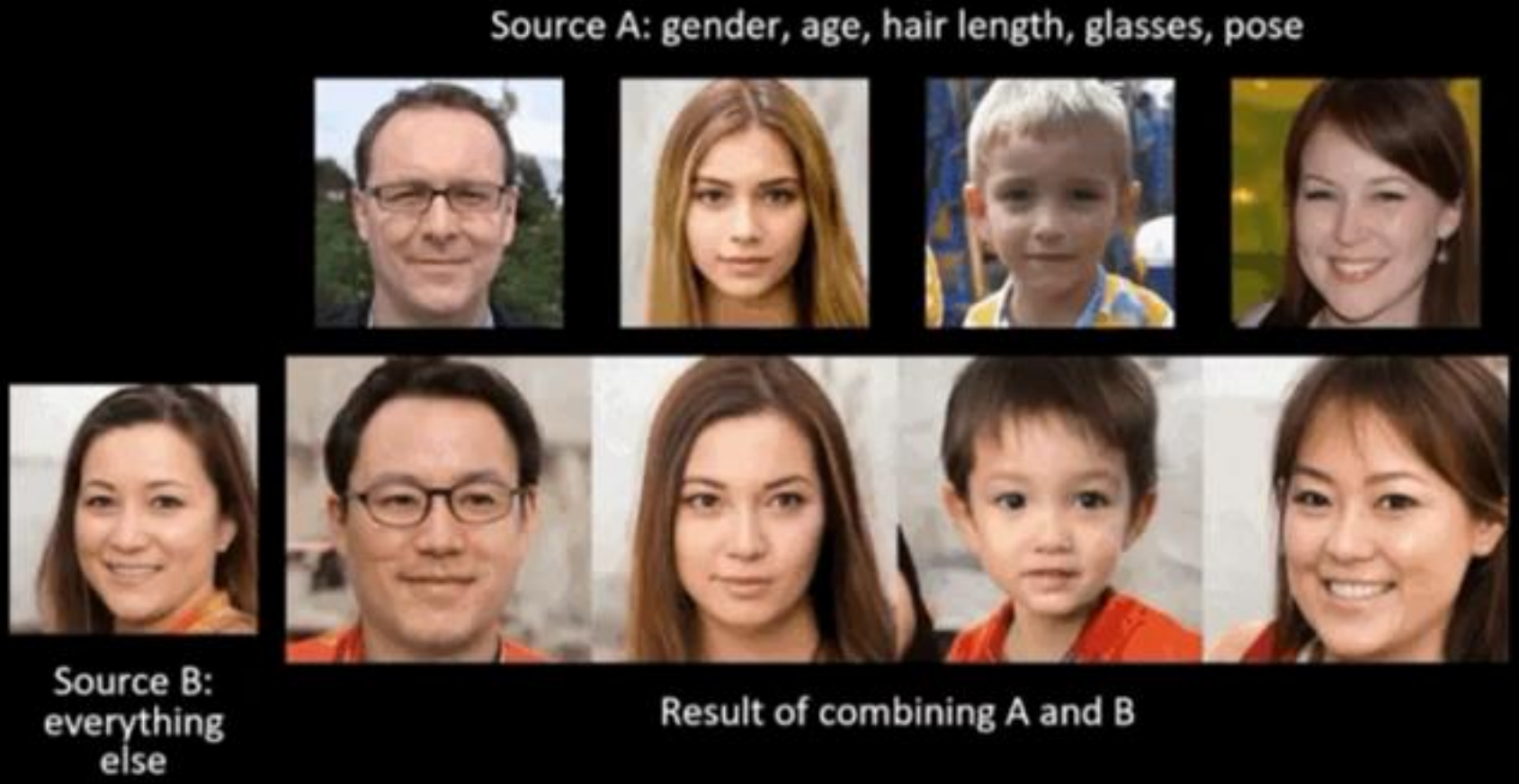


GAN: Generative Adversarial Networks



https://syncedreview.com/2019/02/09/nvidia-open-sources-hyper-realistic-face-generator-stylegan/

GAN: Generative Adversarial Networks



https://syncedreview.com/2019/02/09/nvidia-open-sources-hyper-realistic-face-generator-stylegan/

Image Classification vs. Image Detection

a)

Input X-ray Image

Image Classification → 'Handgun'

b)

Input X-ray Image

Image Detection →

Object	x	y	w	h
Handgun	255	410	510	820
Shuriken	748	405	323	505
Knife	680	850	550	180

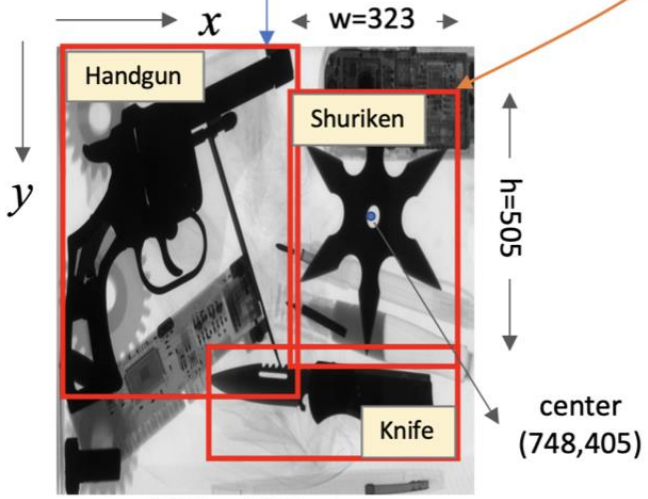


Image Detection: Sliding Windows

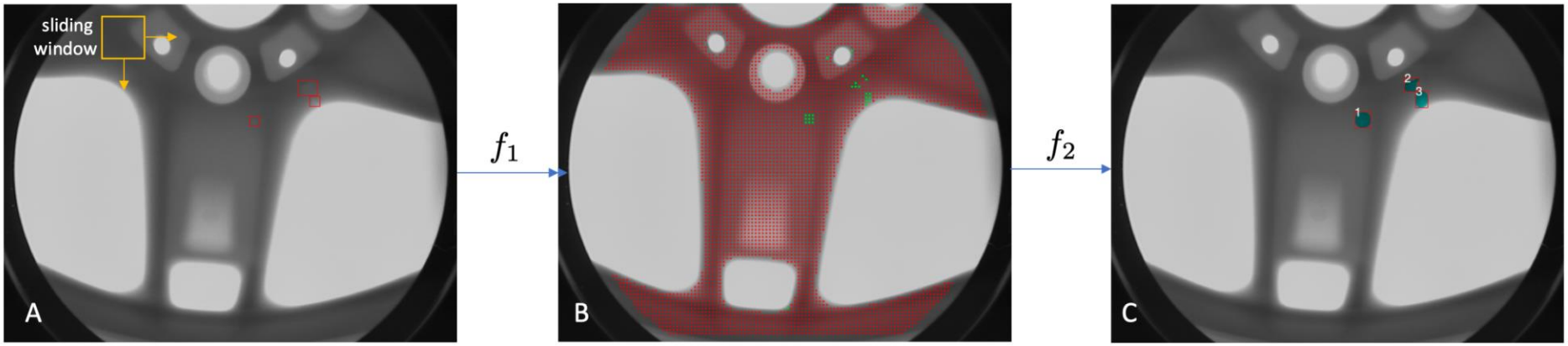


Image Detection: R-CNN

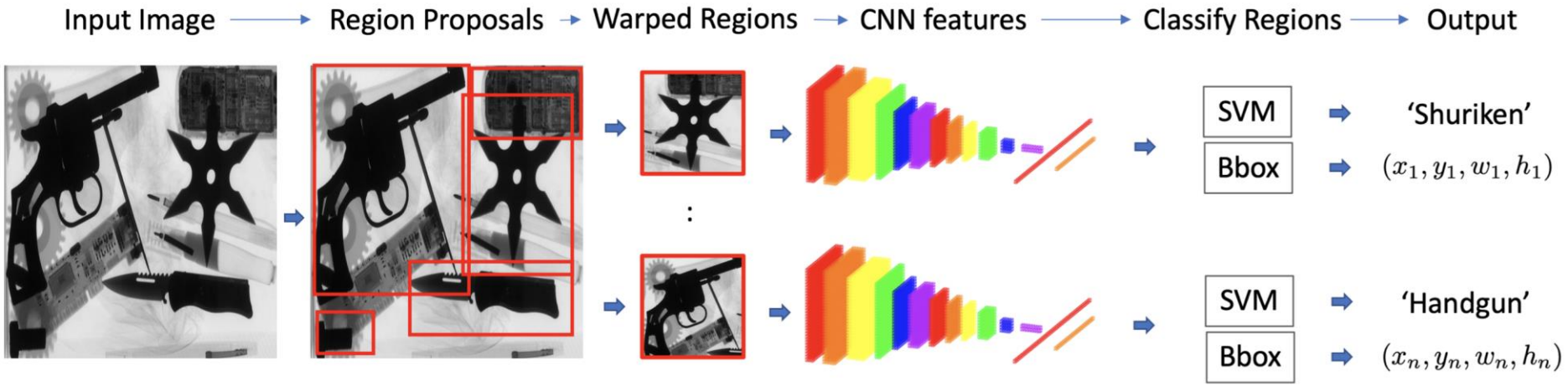


Image Detection: Fast R-CNN

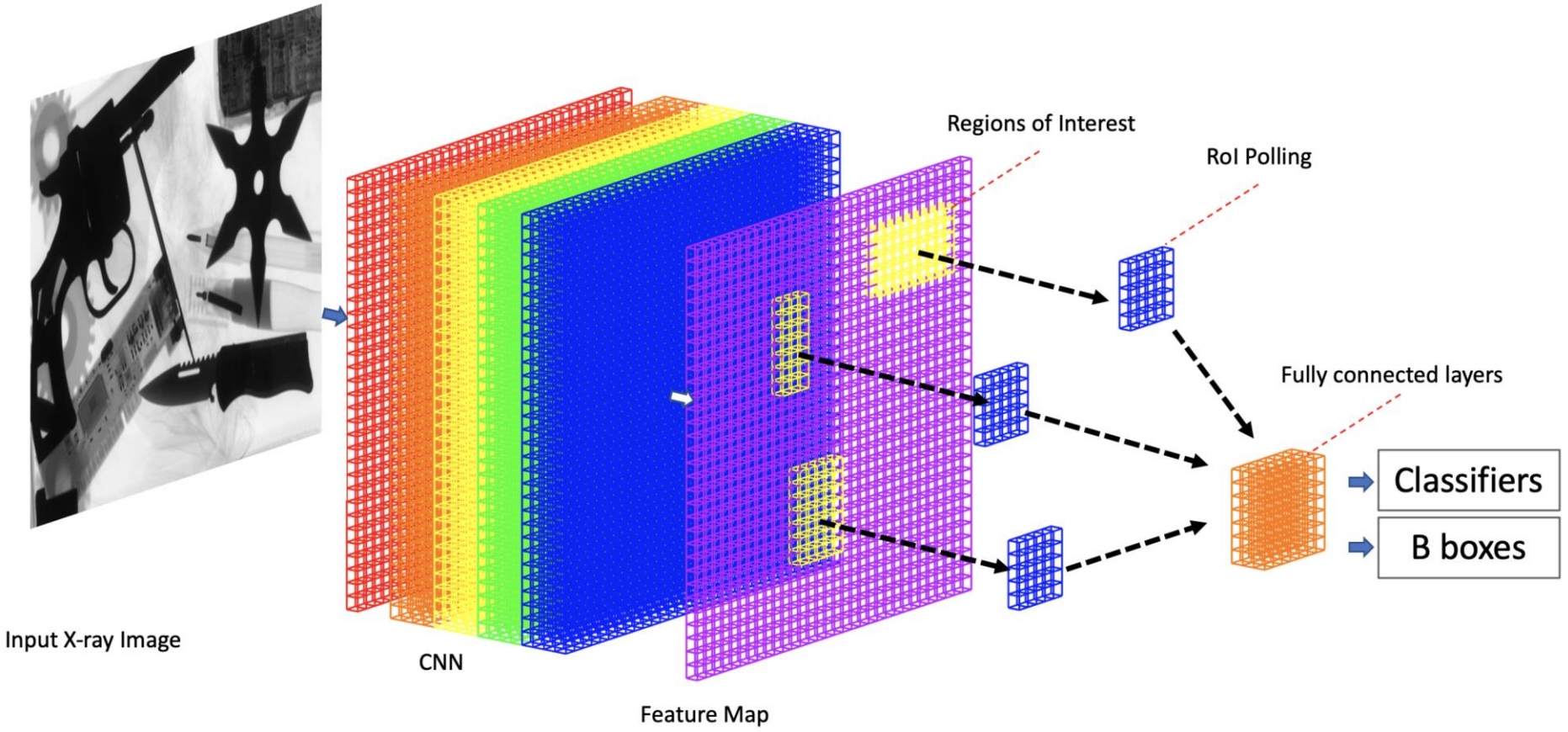


Image Detection: Faster R-CNN and Mask R-CNN

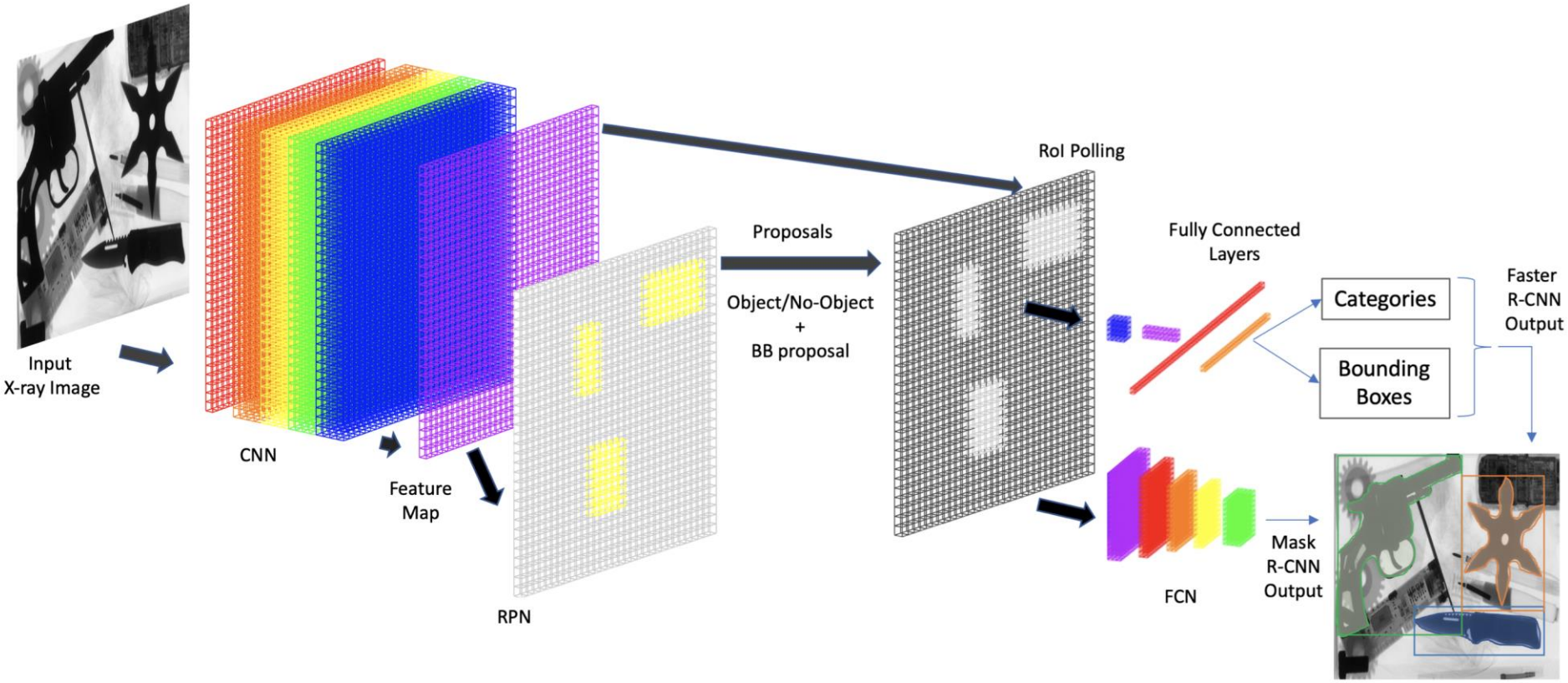
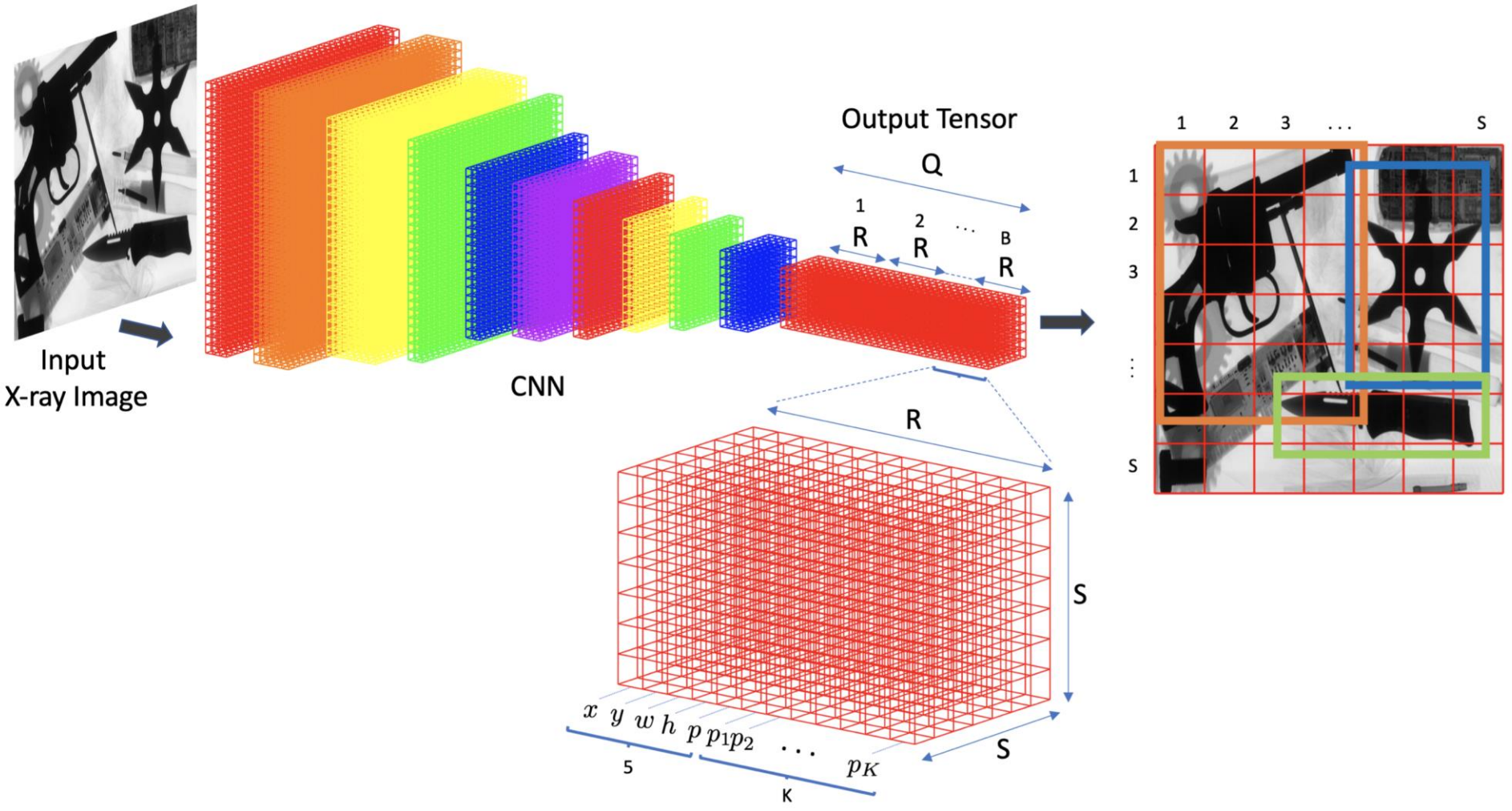
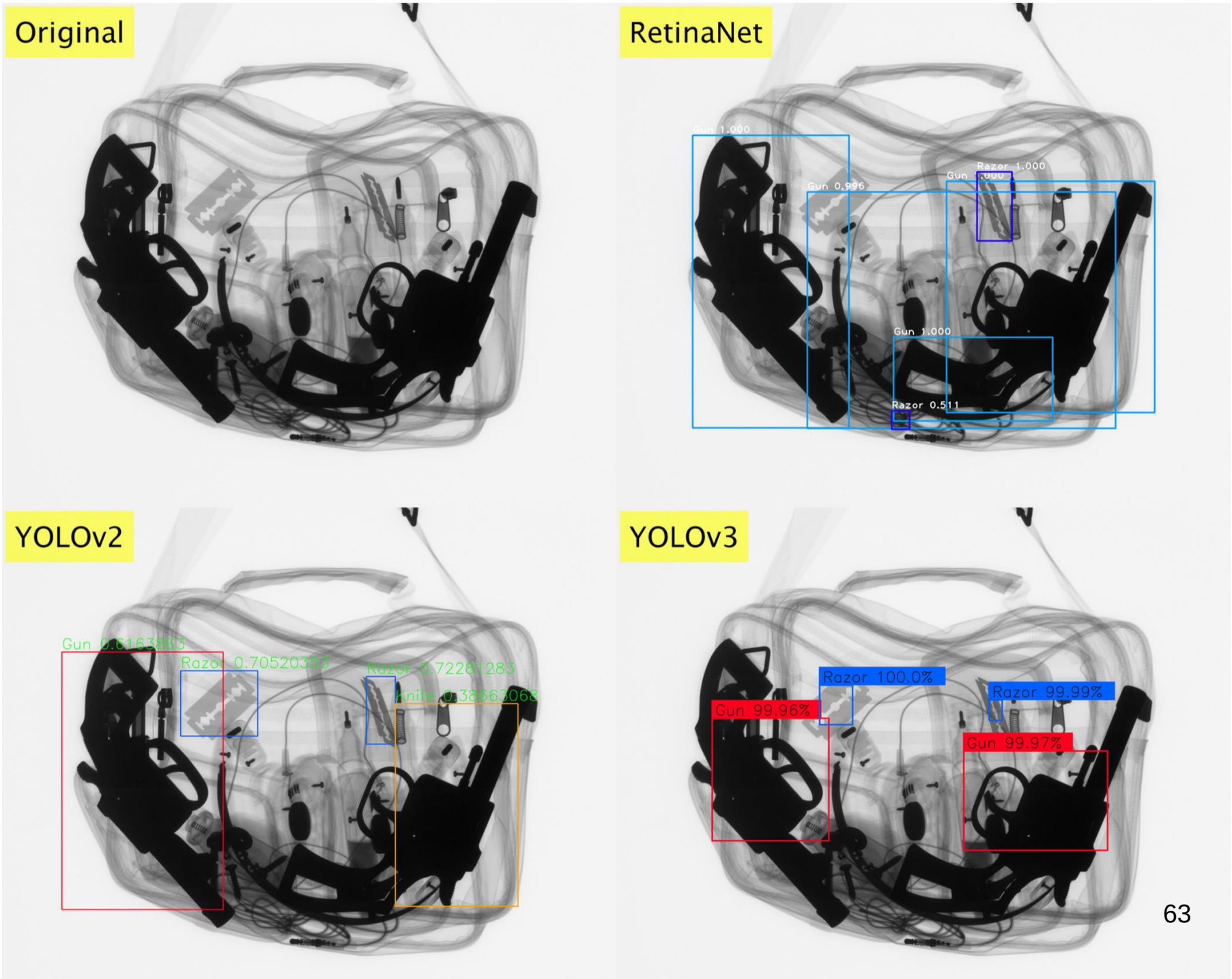


Image Detection: YOLO




Original
RetinaNet
Gun 1.000
Gun 0.996
Razor 1.000
Gun 1.000
Gun 1.000
Razor 0.511
YOLOv2
Gun 0.6163863
Razor 0.70520353
Razor 0.72281283
Knife 0.38663068
YOLOv3
Razor 100.0%
Gun 99.96%
Razor 99.99%
Gun 99.97%


Original
RetinaNet
Razor 1.000
Gun 1.000
Shuriken 1.000
Shuriken 1.000
Gun 1.000
Knife 1.000
YOLOv2
Razor 0.7622877
Shuriken 0.5945094
Shuriken 0.6276606
YOLOv3
Razor 99.96%
Razor 100.0%
Shuriken 99.99%
Shuriken 99.9%