CNN Anatomy and Training

Giacomo Boracchi,
DEIB, Politecnico di Milano
Artificial Neural Networks and Deep Learning AY2023-2024

giacomo.boracchi@polimi.it
https://boracchi.faculty.polimi.it/

Parameters in a CNN

Convolutional Layers

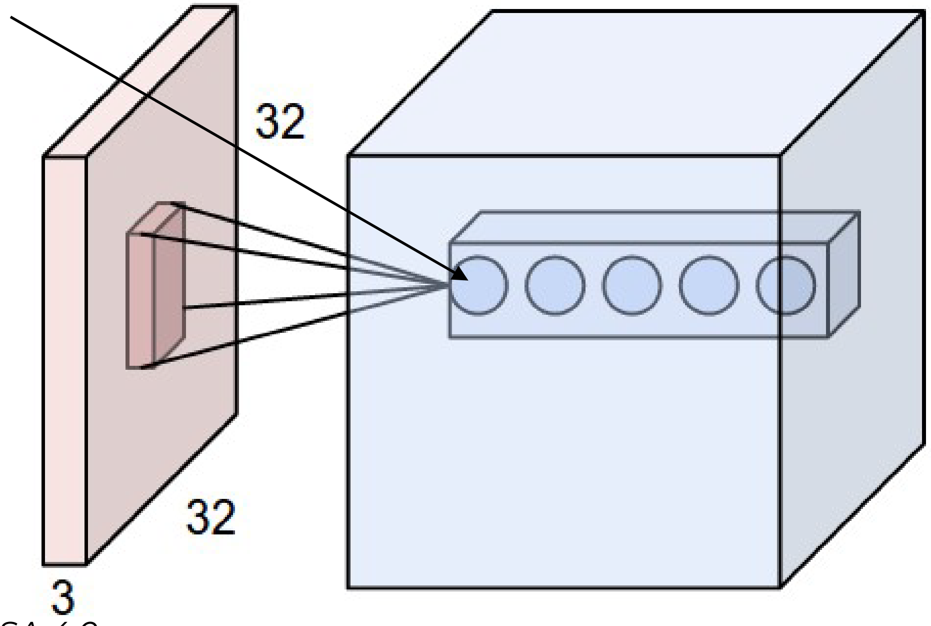
Convolutional layers "mix" all the input components

The output is a linear combination of **all the values in a region of the input, considering all the channels**

$$a(r,c,1) = \sum_{i,j,k} \boxed{w^1(i,j,k)}\, x(r+i, c+j, k) + \boxed{b^1}$$

The parameters of this layer are **called filters.**

The same filter is used through the whole spatial extent of the input



Convolutions as MLP

Convolution is a linear operation!

Therefore, if you unroll the input image to a vector, **you can consider convolution weights as the weights of a Multilayer Perceptron Network!**

inputs

$$a(x,y,k) = \sum_{u,v,z} \boxed{w_k(u,v,z)}\ \boxed{I(x-u, y-v, z)} + \boxed{b_k}$$

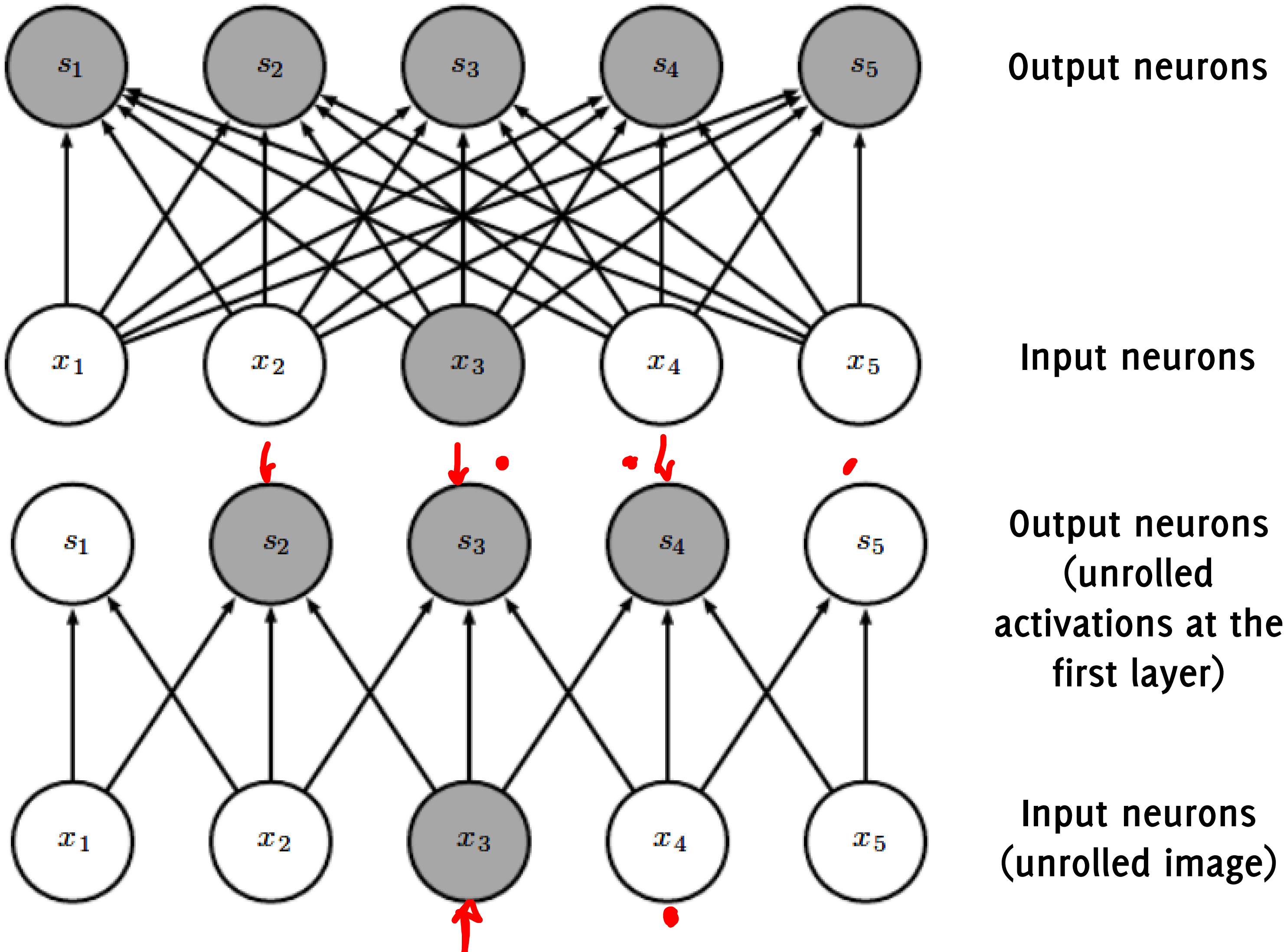
parameters

What are the differences between MLP and CNNs then?

CNNs has Sparse Connectivity

MLP, Fully connected

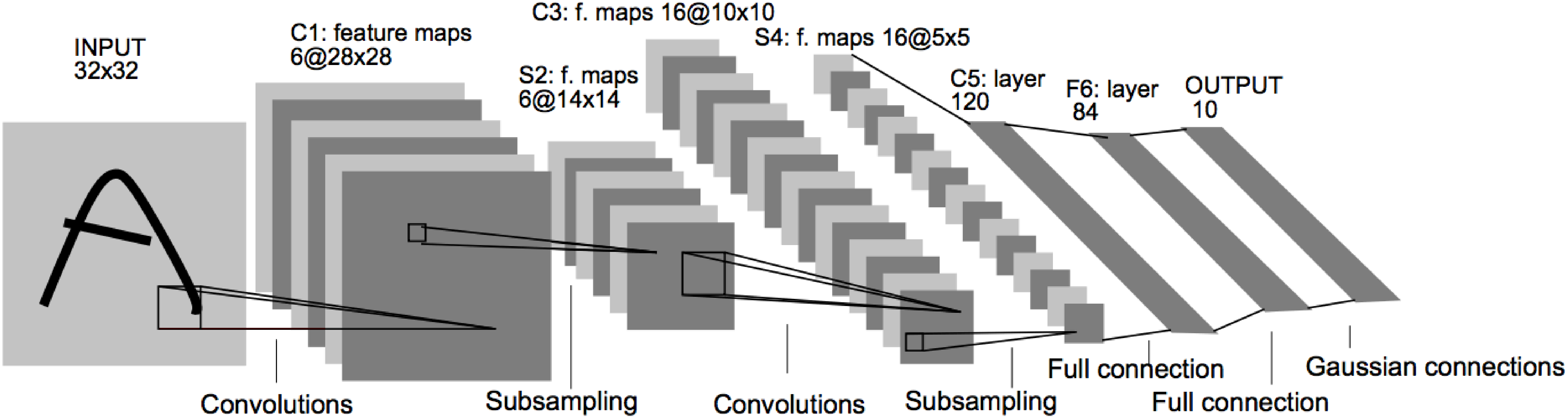
3x1 convolutional



Weight Sharing / Spatial Invariance

In a CNN, all the neurons in the same slice of a feature map use the same weights and bias: this reduces the nr. of parameters in the CNN.

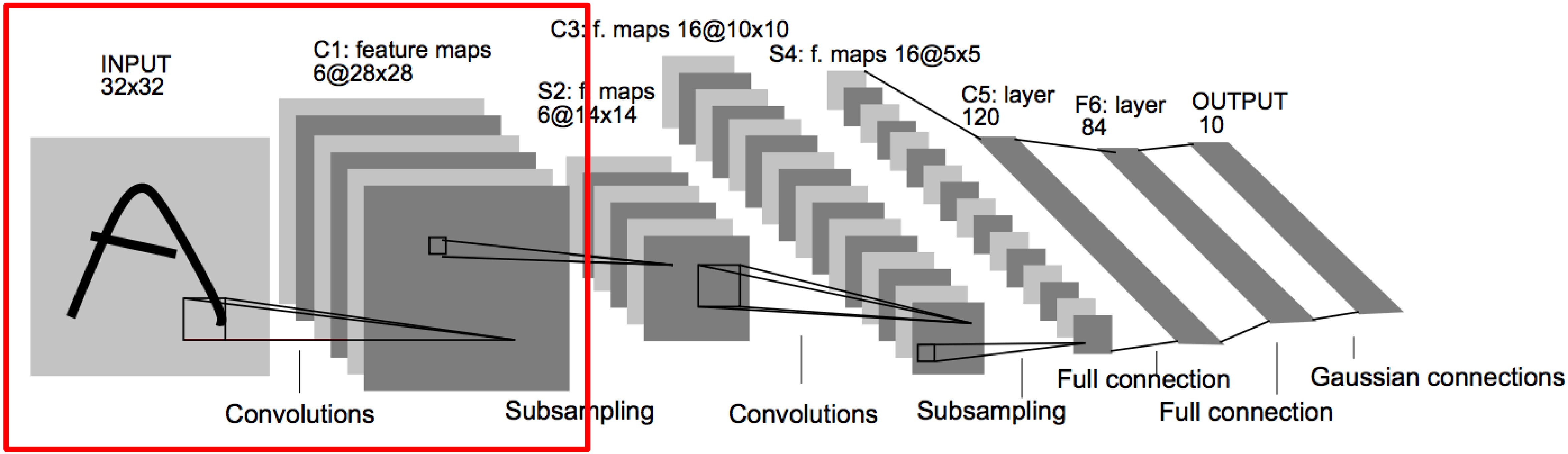
Underlying assumption: **if one feature is useful** to compute **at some spatial position** (x, y), **then it should also be useful** to compute at a **different position** (x_2, y_2)



Weight Sharing / Spatial Invariance

If the first layer were a MLP:

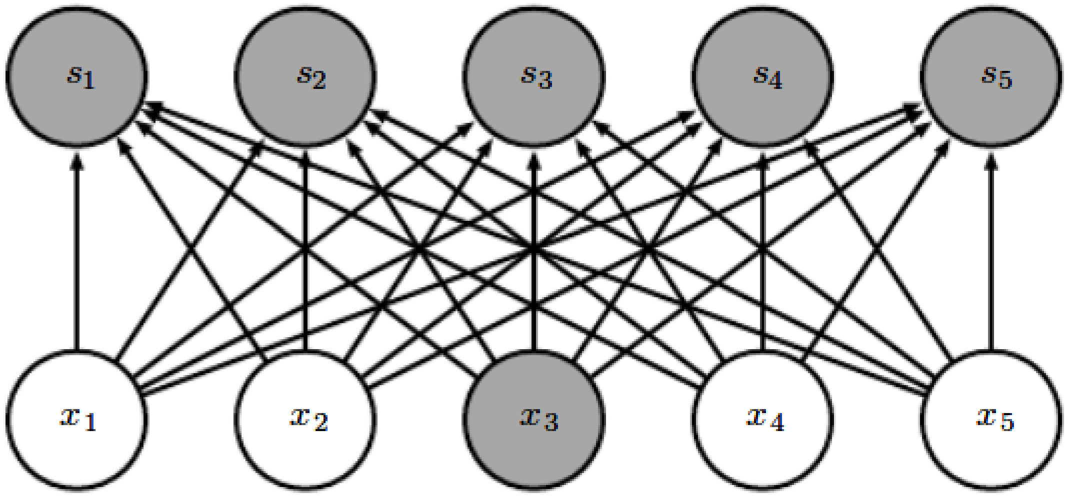
- MLP layer: this should have had 28 x 28 x 6 neurons in the output
- MLP layer with sparse connectivity: only 5x5 nonzero weights each neuron
- MLP layer: 28 x 28 x 6 x 25 weights + 28 x 28 x 6 biases (122 304)
- **Conv layer: 25 weights + 6 biases shared among neurons of the same layer**

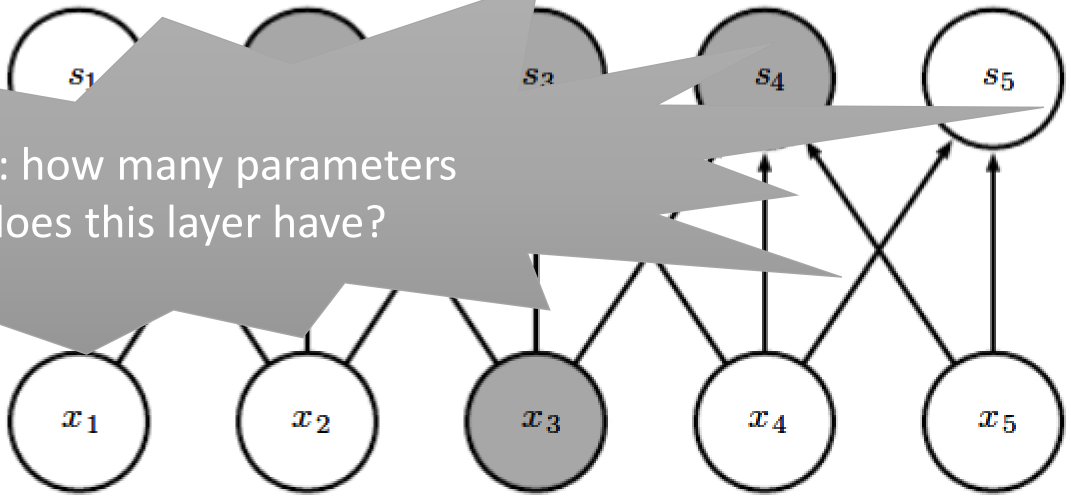


Parameter sharing

Fully connected

5x5 = 25 weights
(+ 5 bias)





Quiz: how many parameters
does this layer have?

3x1 convol

3 weights!
(+ 1 bias)

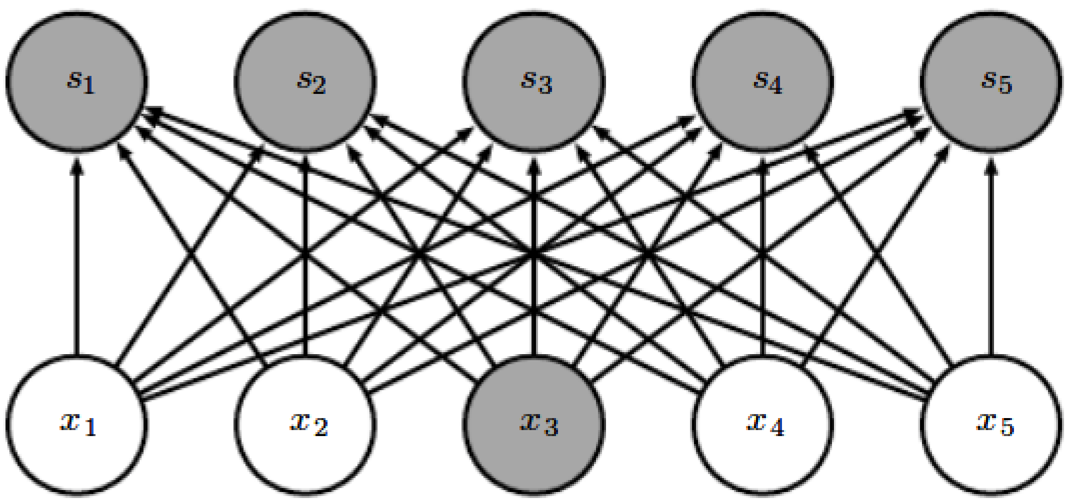
Parameter sharing

+ 5 biases

5 outputs

Fully connected

5x5 = 25 weights
(+ 5 bias)

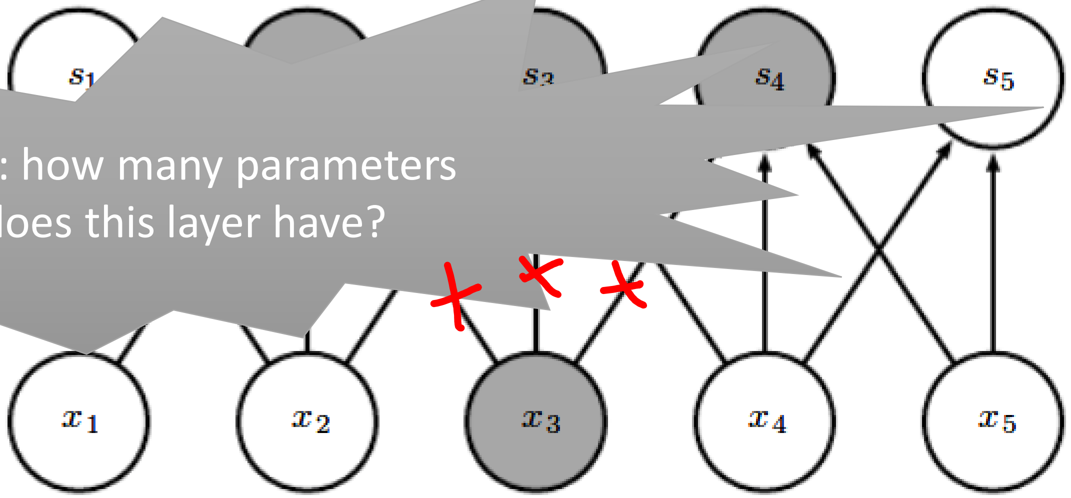


3 inputs

Quiz: how many parameters does this layer have?

3x1 convo

3 weights!
(+ 1 bias)



To Summarize

Any CONV layer can be implemented by a FC layer performing exactly the same computations.

The weight matrix W of the FC layer would be

- a large matrix (#rows equal to the number of output neurons, #cols equal to the nr of input neurons).
- That is mostly zero except for at certain blocks where the local connectivity takes place.
- The weights in many of the blocks are equal due to parameter sharing.

… and we will see that the converse interpretation (FC as conv) is also viable and very useful!

The Receptive Field
A very important aspect in CNNs

The Receptive Field

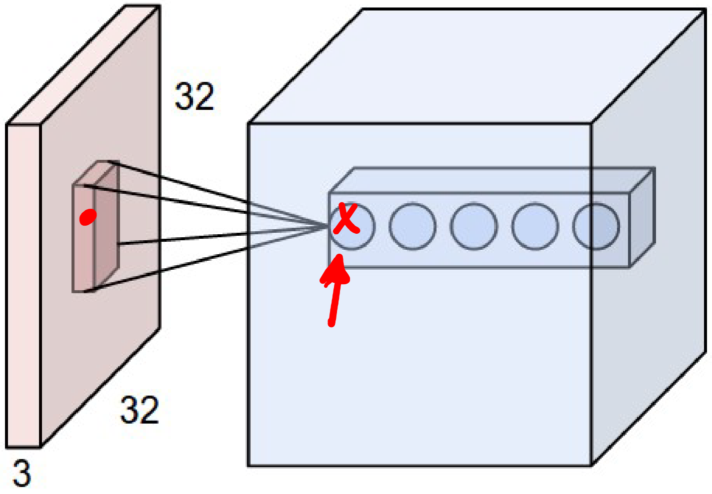
One of the basic concepts in deep CNNs.

Due to sparse connectivity, unlike in FC networks where the value of each output depends on the entire input, **in CNN each output only depends on a specific region in the input.**

This region in the input is the receptive field for that output

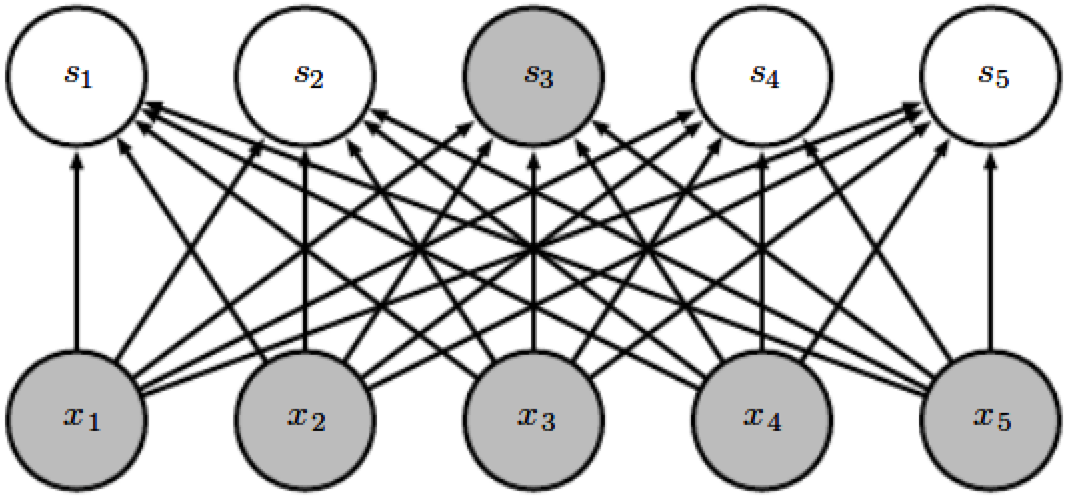
The deeper you go, the wider the receptive field is: maxpooling, convolutions and stride > 1 increase the receptive field

Usually, the receptive field refers to the final **output unit** of the network in relation to the network input, but the same definition holds for intermediate volumes

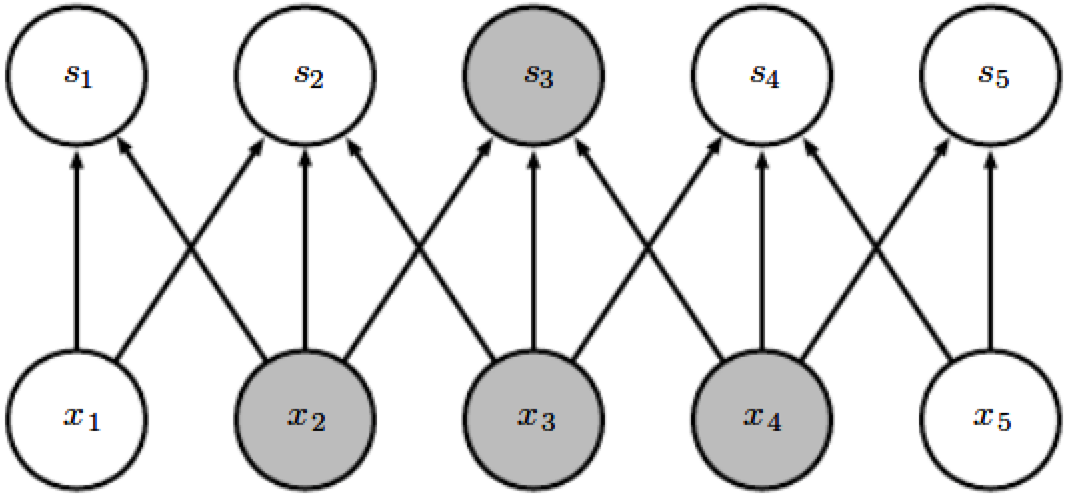


Receptive fields

MPL, Fully connected



3x1 convolutional

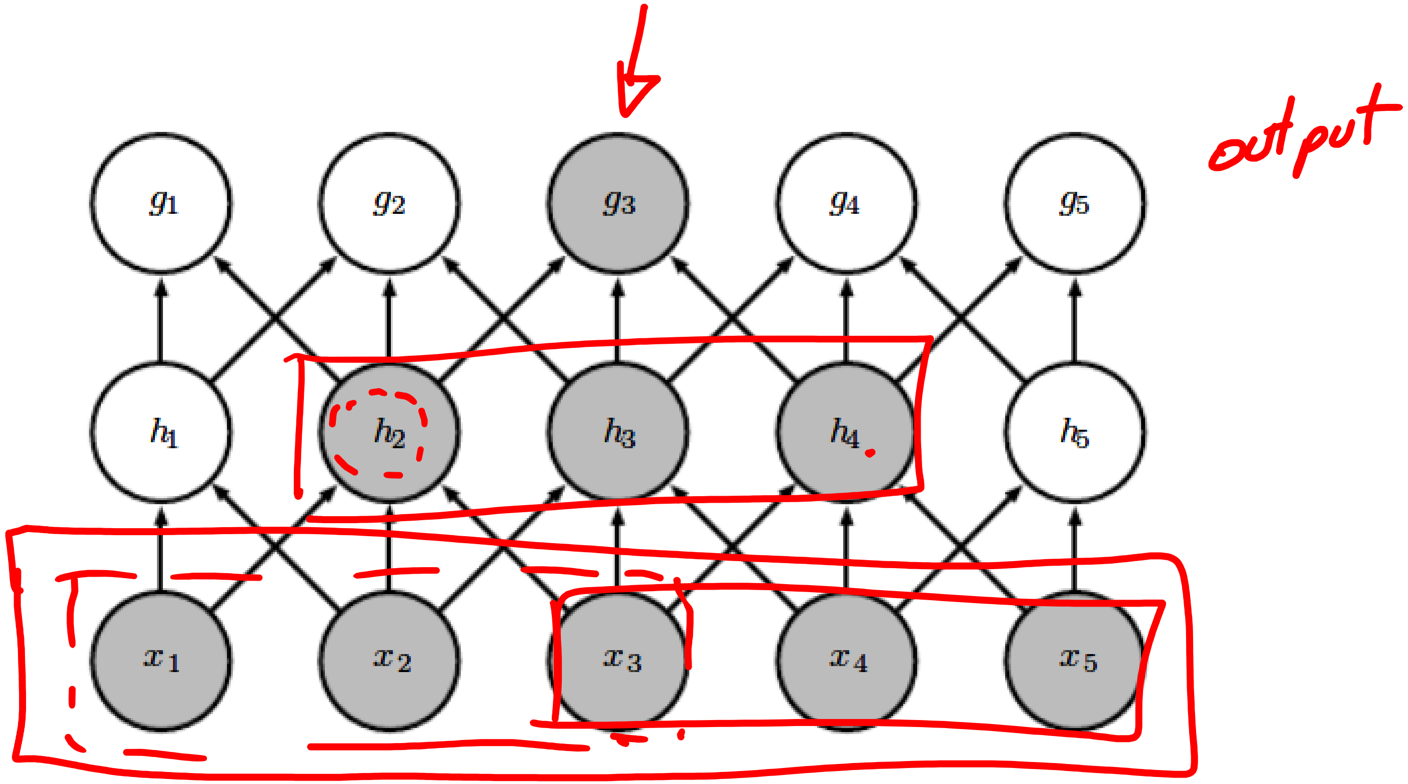


Receptive fields

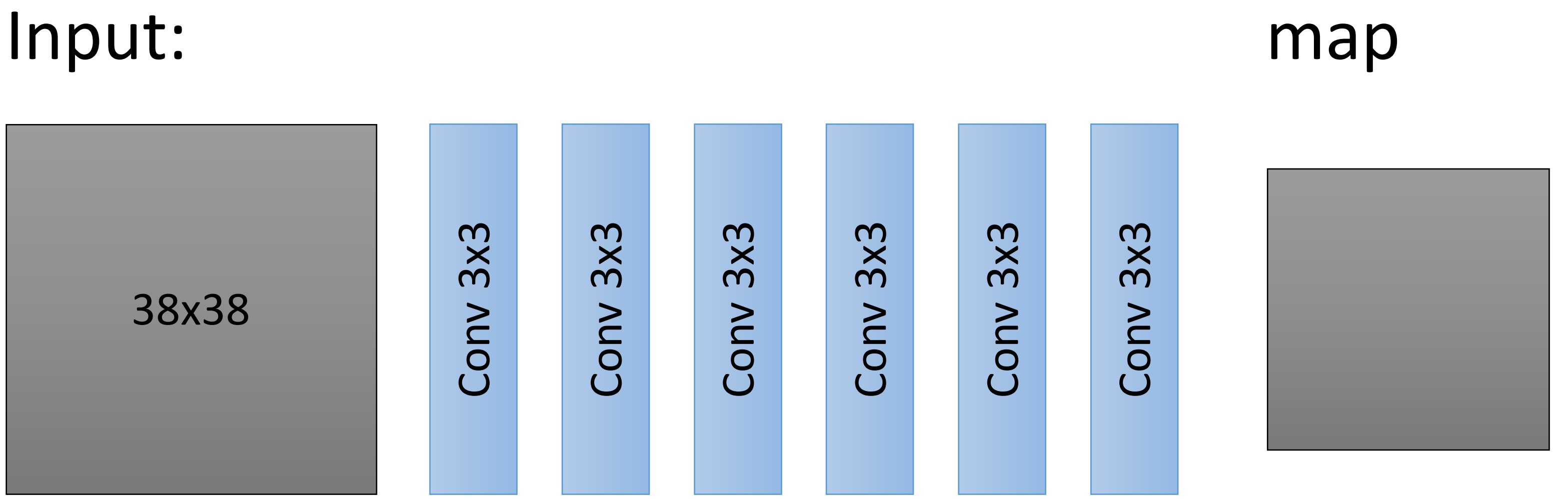
Deeper neurons depend on wider patches of the input (convolution is enough to increase receptive field, no need of maxpooling)

3x1 convolutional

3x1 convolutional

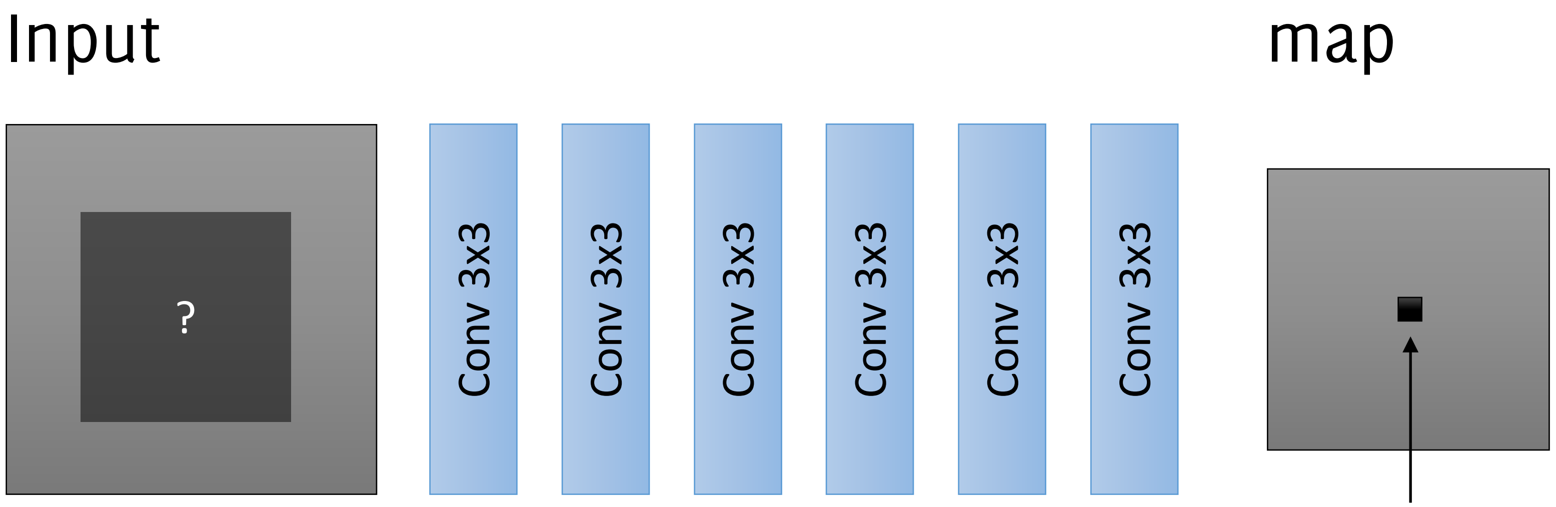


Exercise



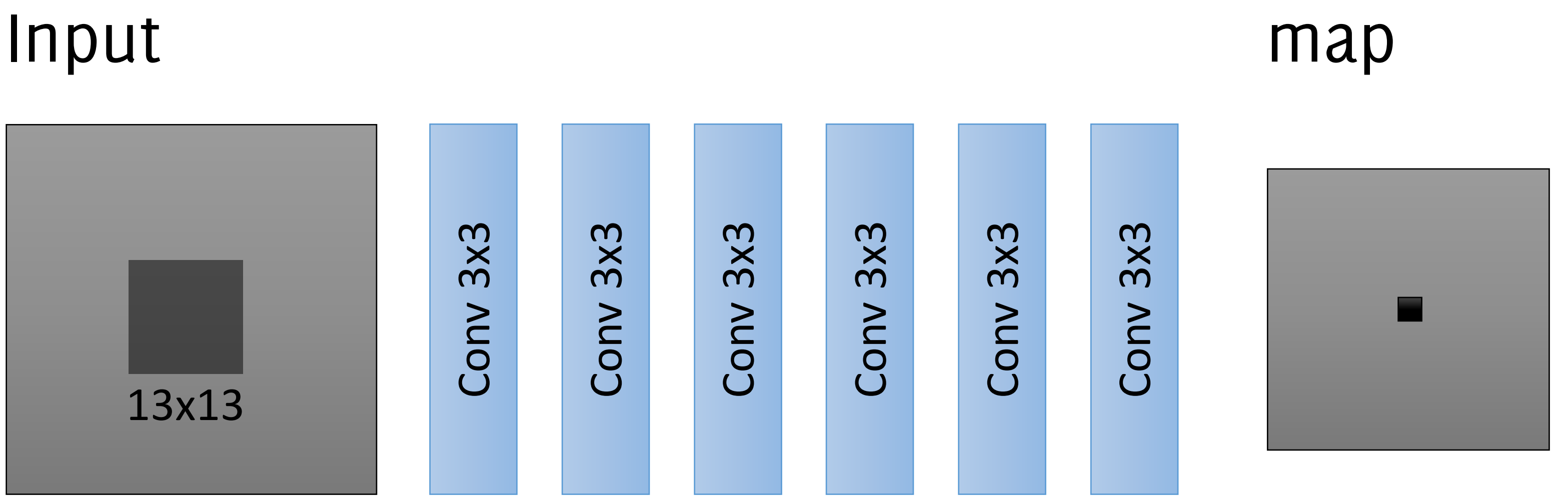
Input:
38x38
Conv 3x3
Conv 3x3
Conv 3x3
Conv 3x3
Conv 3x3
Conv 3x3
map

Receptive fields



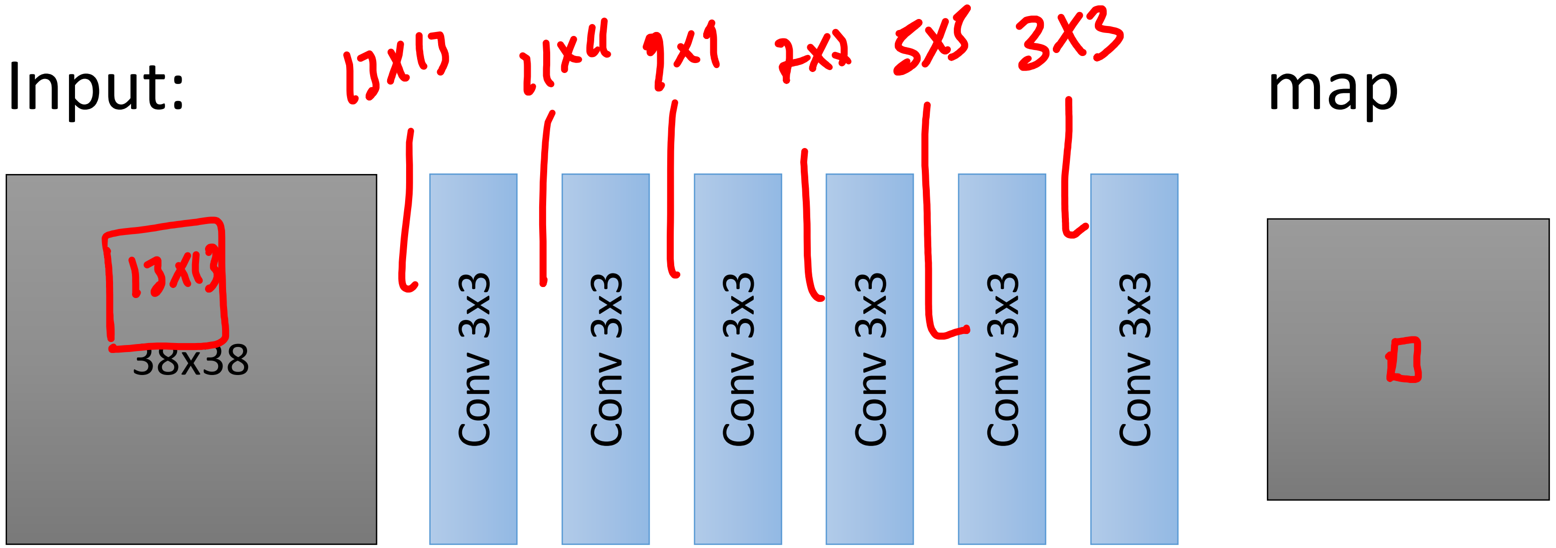
How large is the receptive field of the black neuron?

Receptive fields

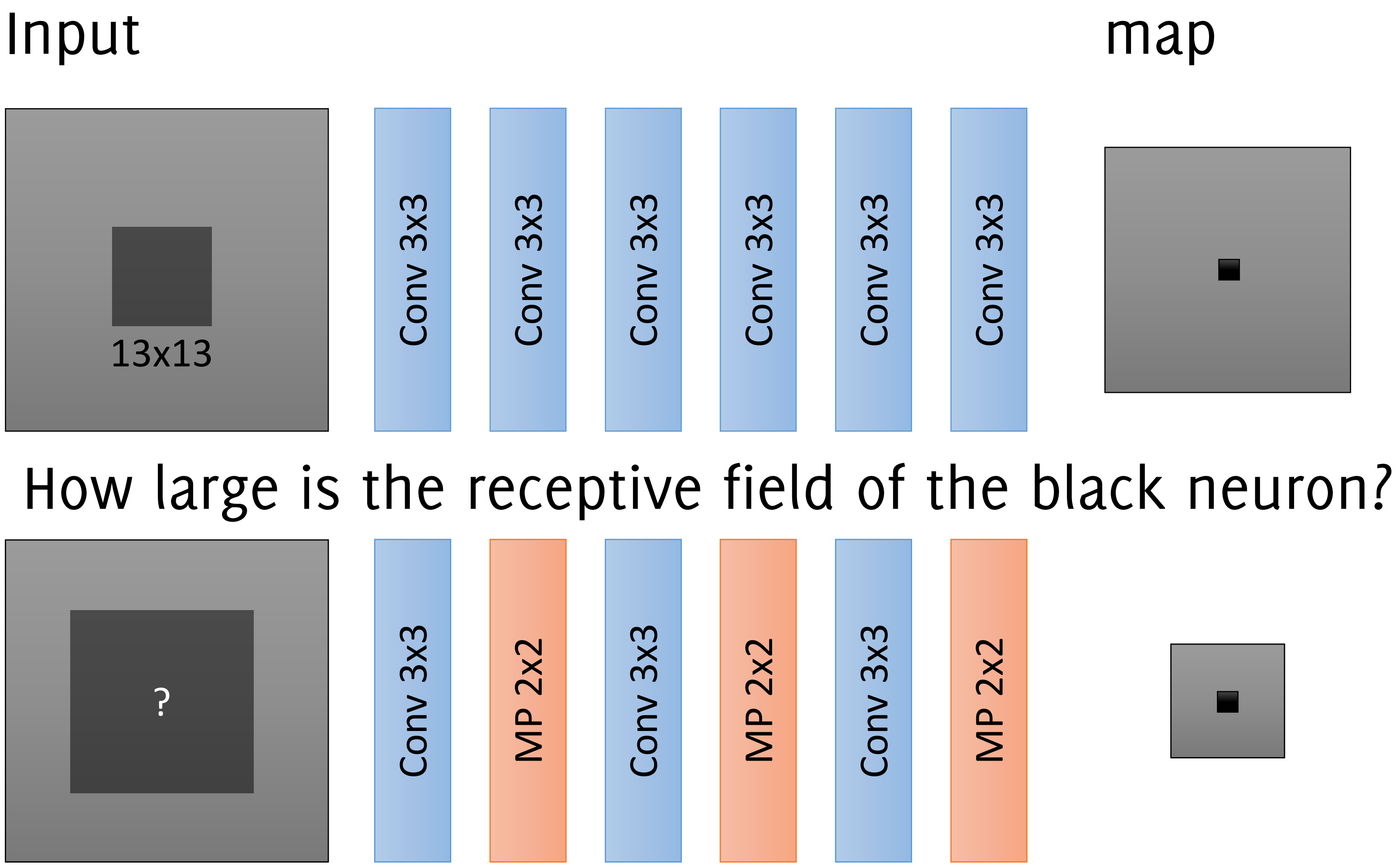


How large is the receptive field of the black neuron?

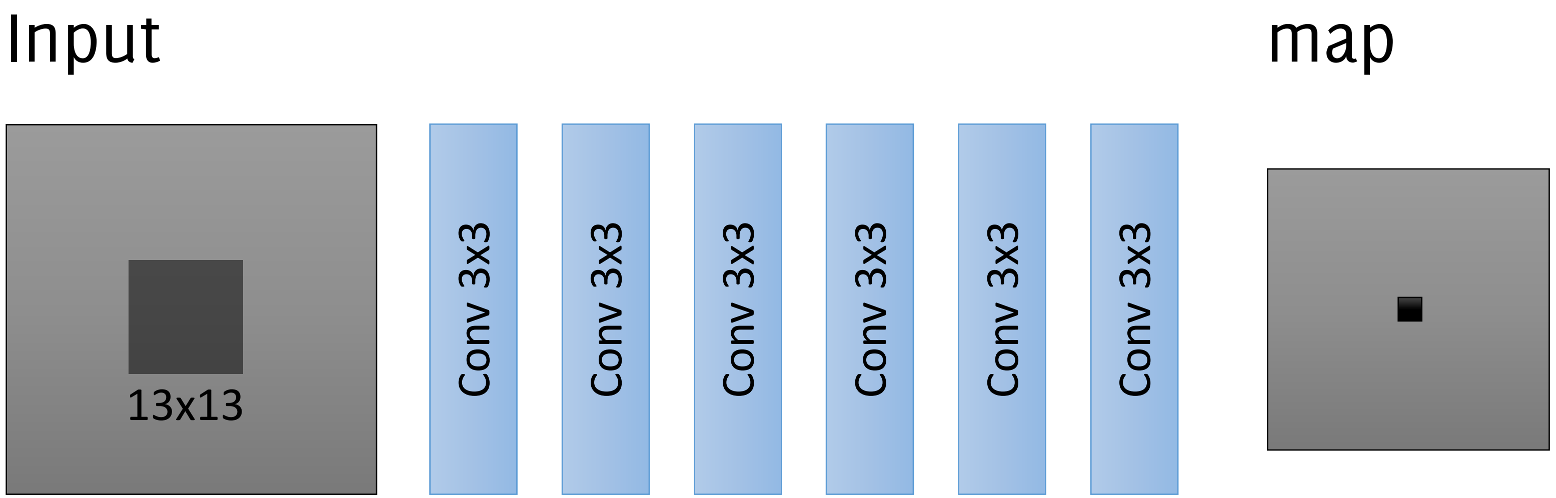
Exercise


Input:
13x13
11x11
9x9
7x7
5x5
3X3
map
13x13
38x38
Conv 3x3
Conv 3x3
Conv 3x3
Conv 3x3
Conv 3x3
Conv 3x3

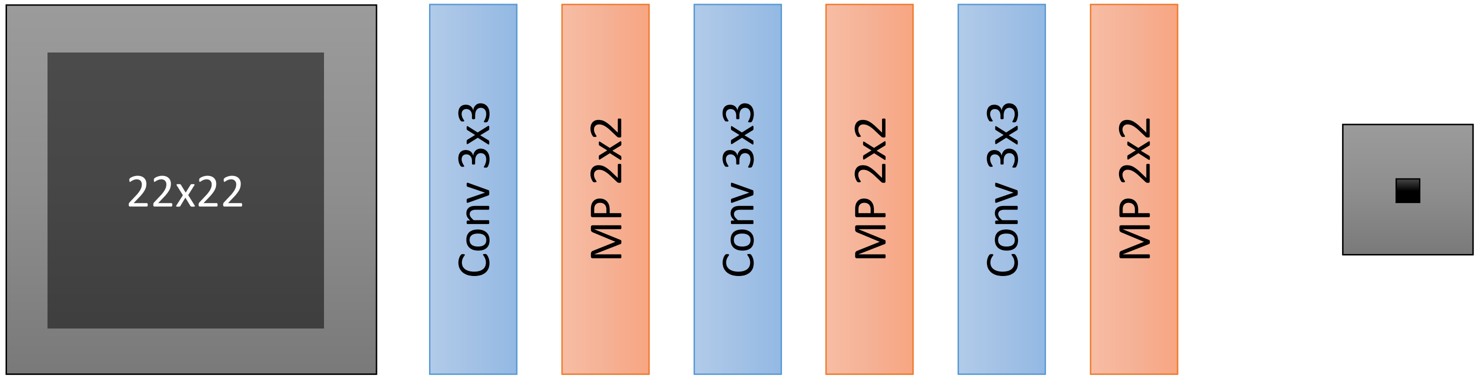
Receptive fields



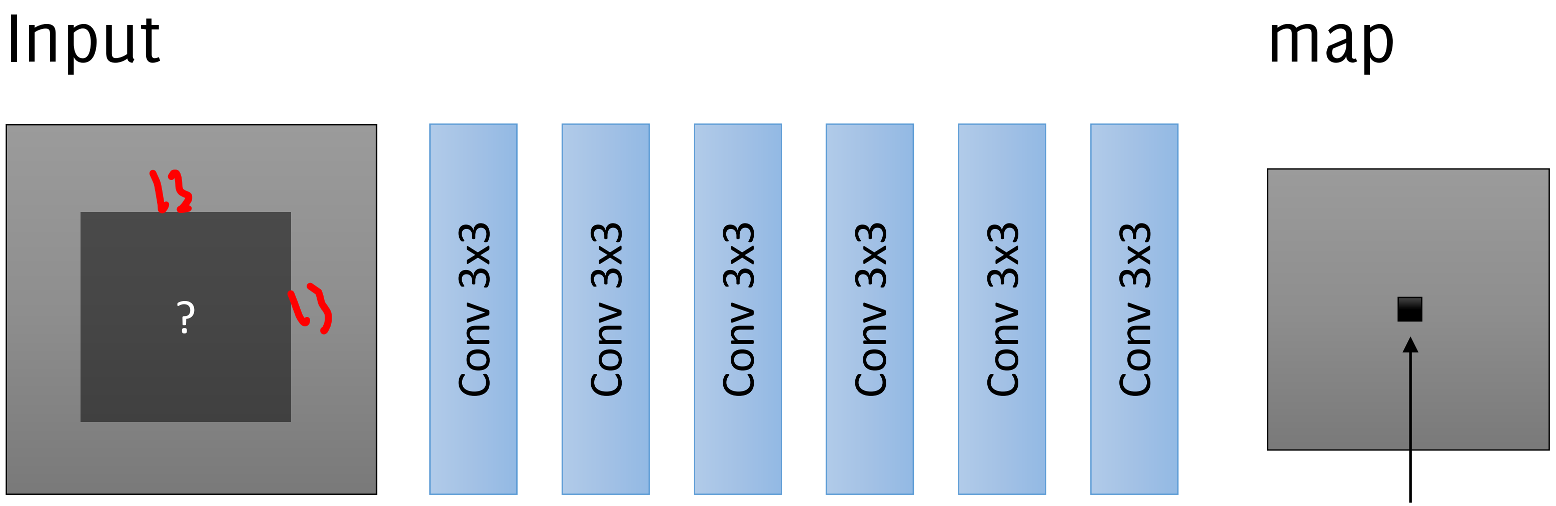
Receptive fields

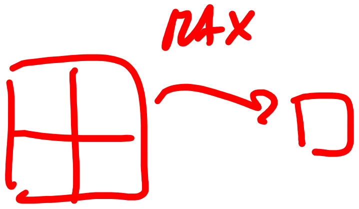


How large is the receptive field of the black neuron?

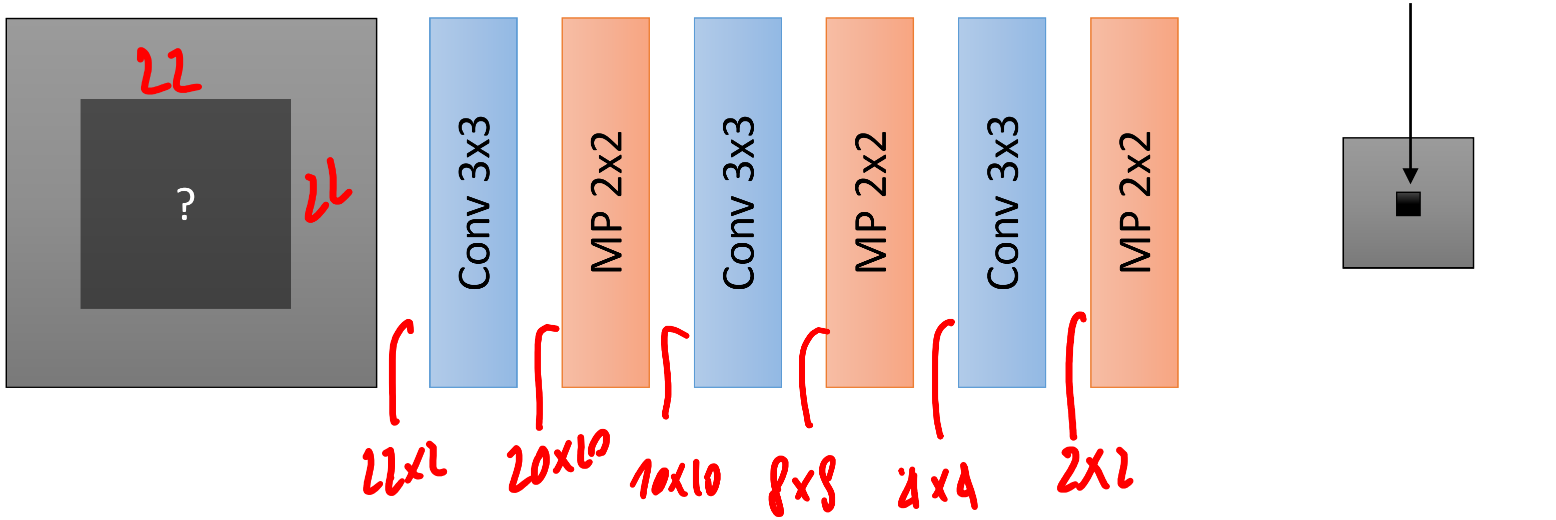


Receptive fields





How large is the receptive field of the black neuron?



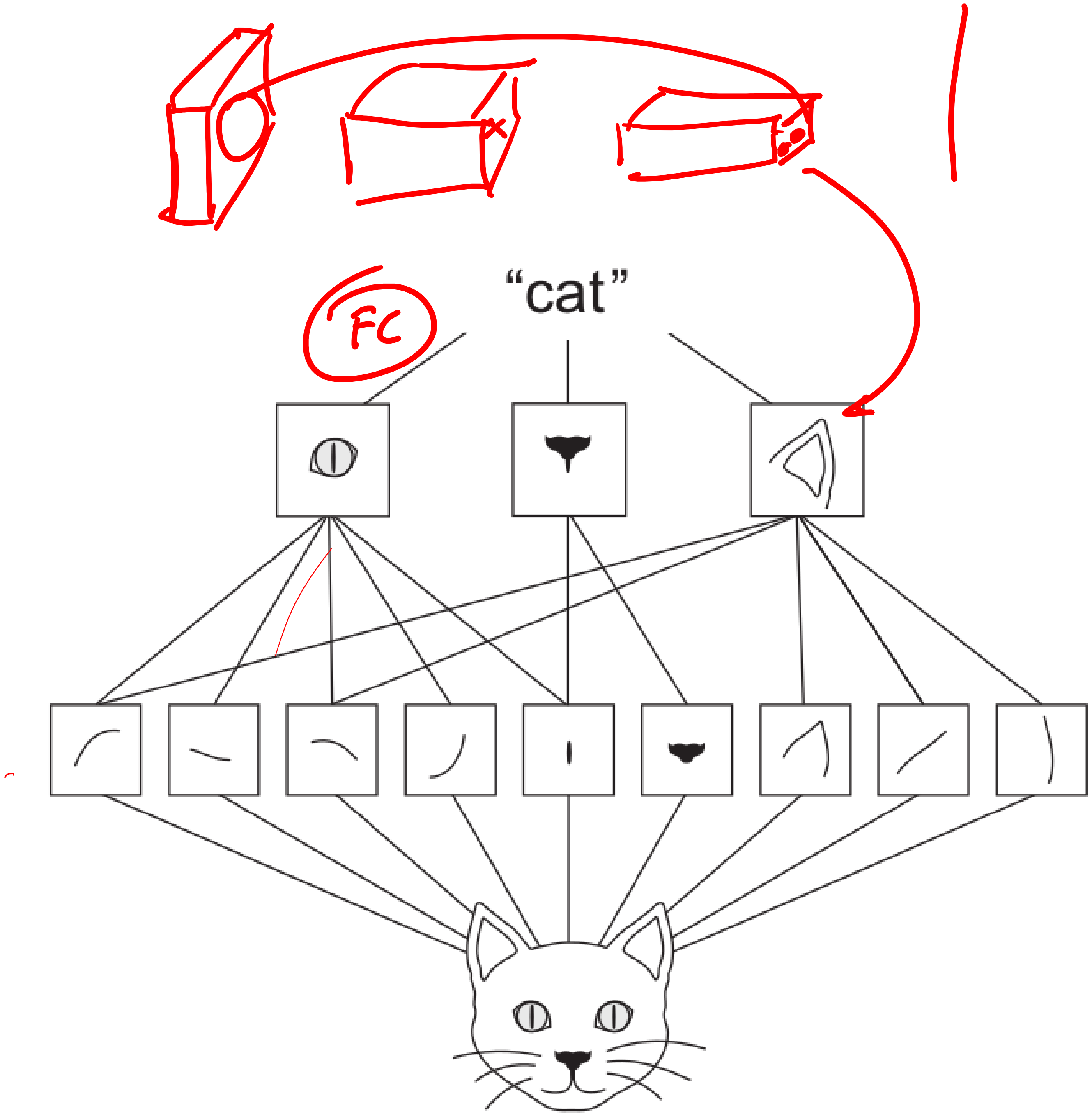
As we move deeper…

As we move to deeper layers:

- spatial resolution is reduced
- the number of maps increases

We search for **higher-level patterns**, and **don't care too much about their exact location.**

There are more high-level patterns than low-level details!



CNN Training

Training a CNN

- **Each CNN can be seen as a MLP**, with sparse and shared connectivities)
- CNN can be in principle **trained by gradient descent** to minimize a loss function over a batch (e.g. binary cross-entropy, RMSE, Hinge loss..)
- Gradient can be computed by **backpropagation** (chain rule) as long as we can derive each layer of the CNN
- **Weight sharing needs to be taken into account** (fewer parameters to be used in the derivatives) while computing derivatives
- There are just a few details missing…

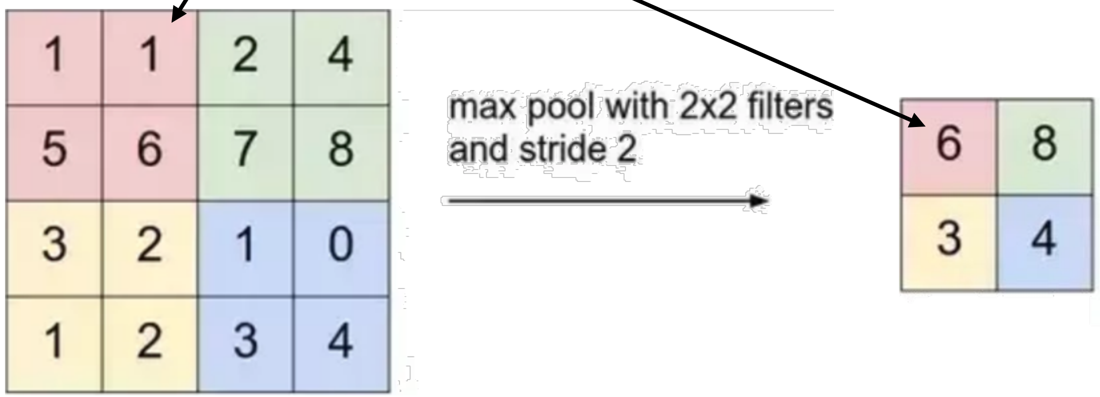
Detail: backprop with max pooling

The gradient is only routed through the input pixel that contributes to the output value; e.g.:

Gradient of • with respect to • = 0

The derivative is:

- 1 at the location corresponding the maximum
- 0 otherwise



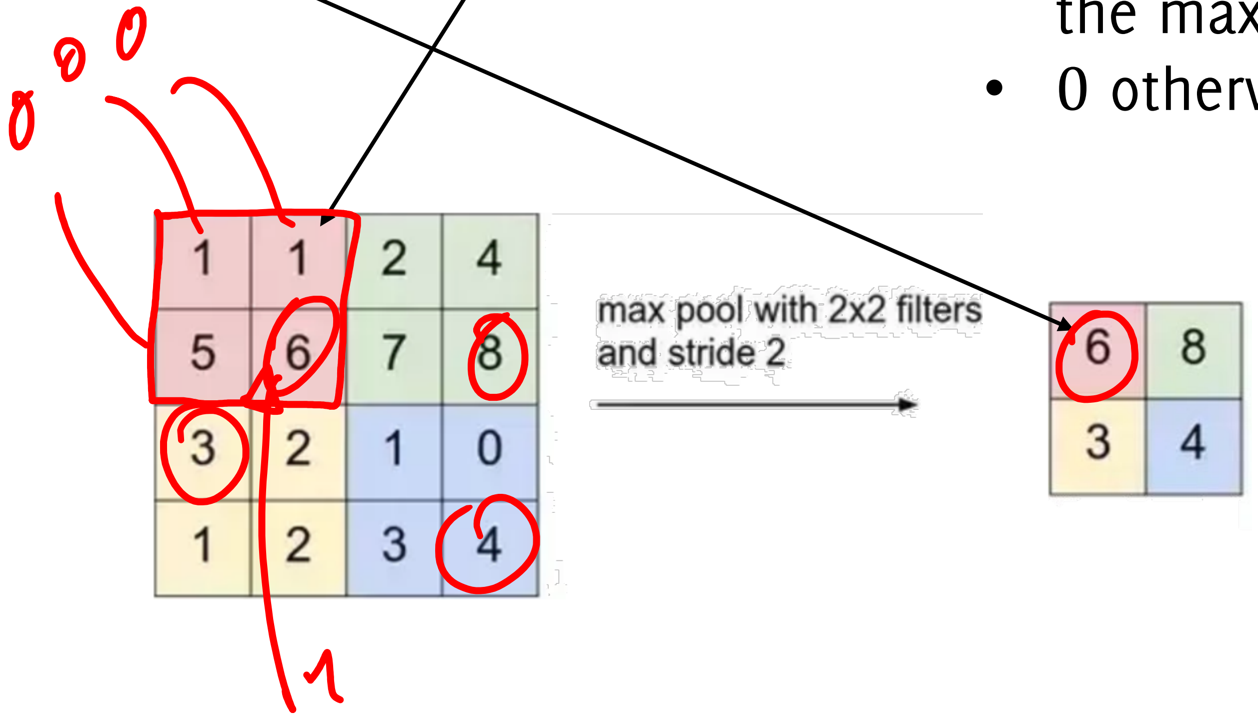
Detail: backprop with max pooling

The gradient is only routed through the input pixel that contributes to the output value; e.g.:

Gradient of • with respect to • = 0

The derivative is:

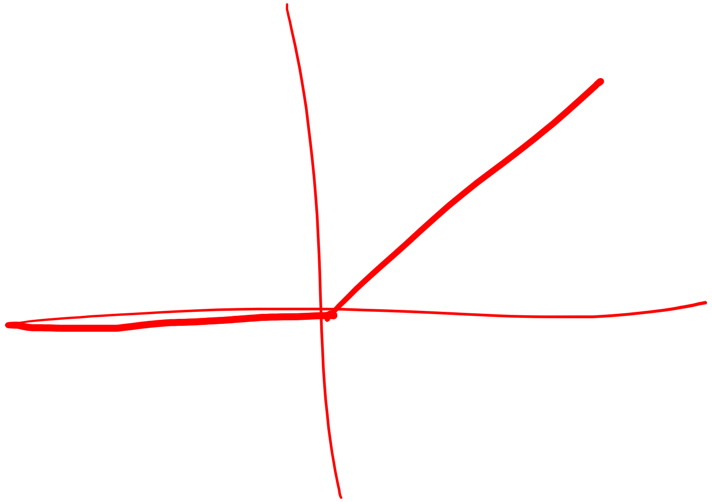
- 1 at the location corresponding the maximum
- 0 otherwise



Detail: derivative of ReLU

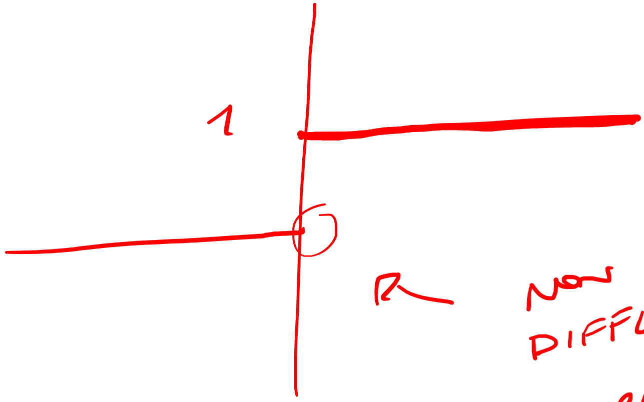
The ReLU derivative is straightforward

RELU



$$\begin{cases} x & x \geq 0 \\ 0 & x < 0 \end{cases}$$

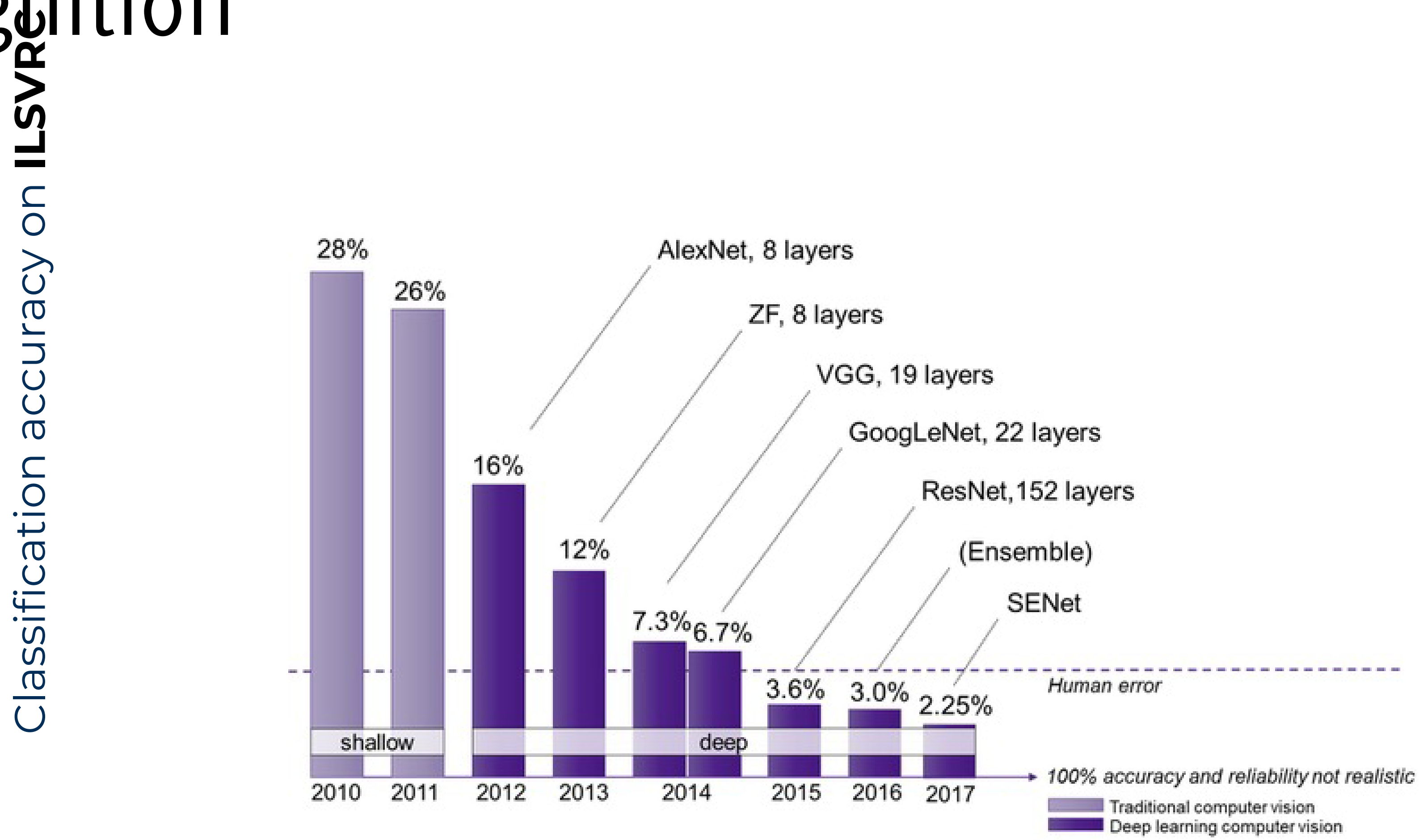
DERIVATIVE



NON DIFFERENTIABLE, SUBGRADIENT, SET THIS TO 0

A Breaktrough in Image Classification

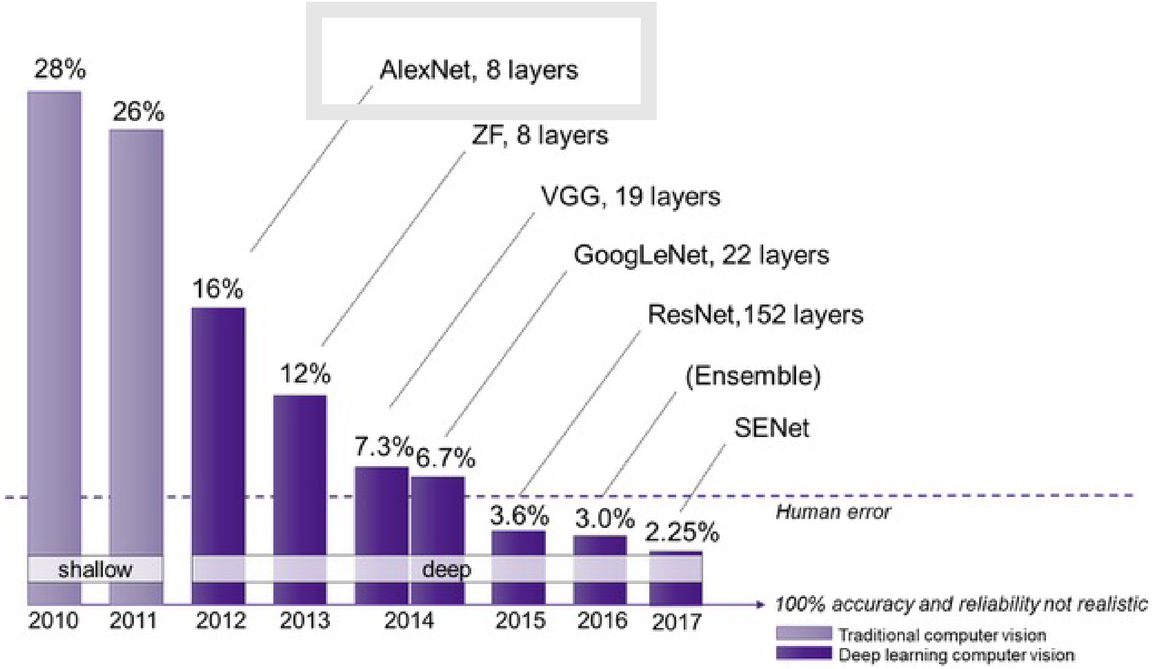
The impact of Deep Learning in Visual Recognition

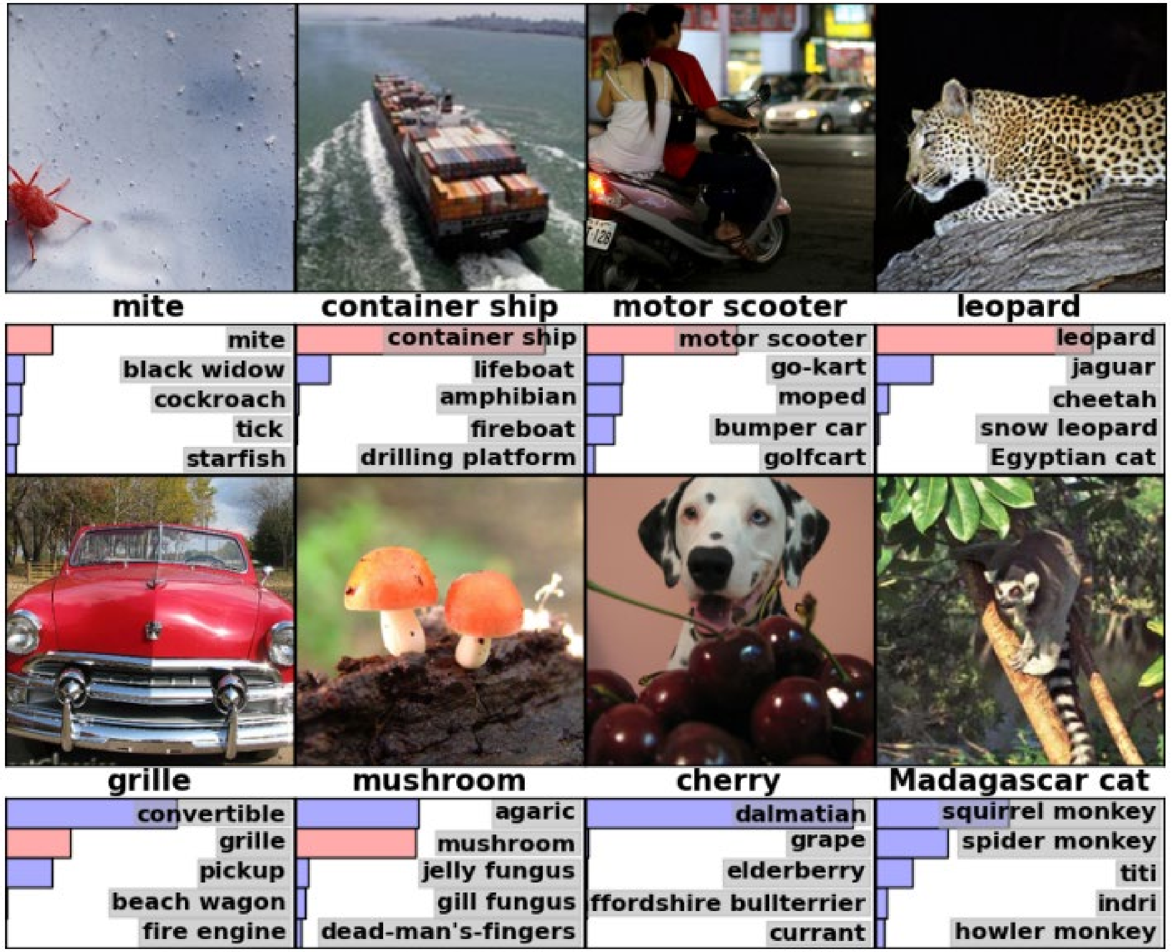


Many layers!

ILSVRC: ImageNet Large Scale Visual Recognition Challenge

AlexNet / Imagenet Images

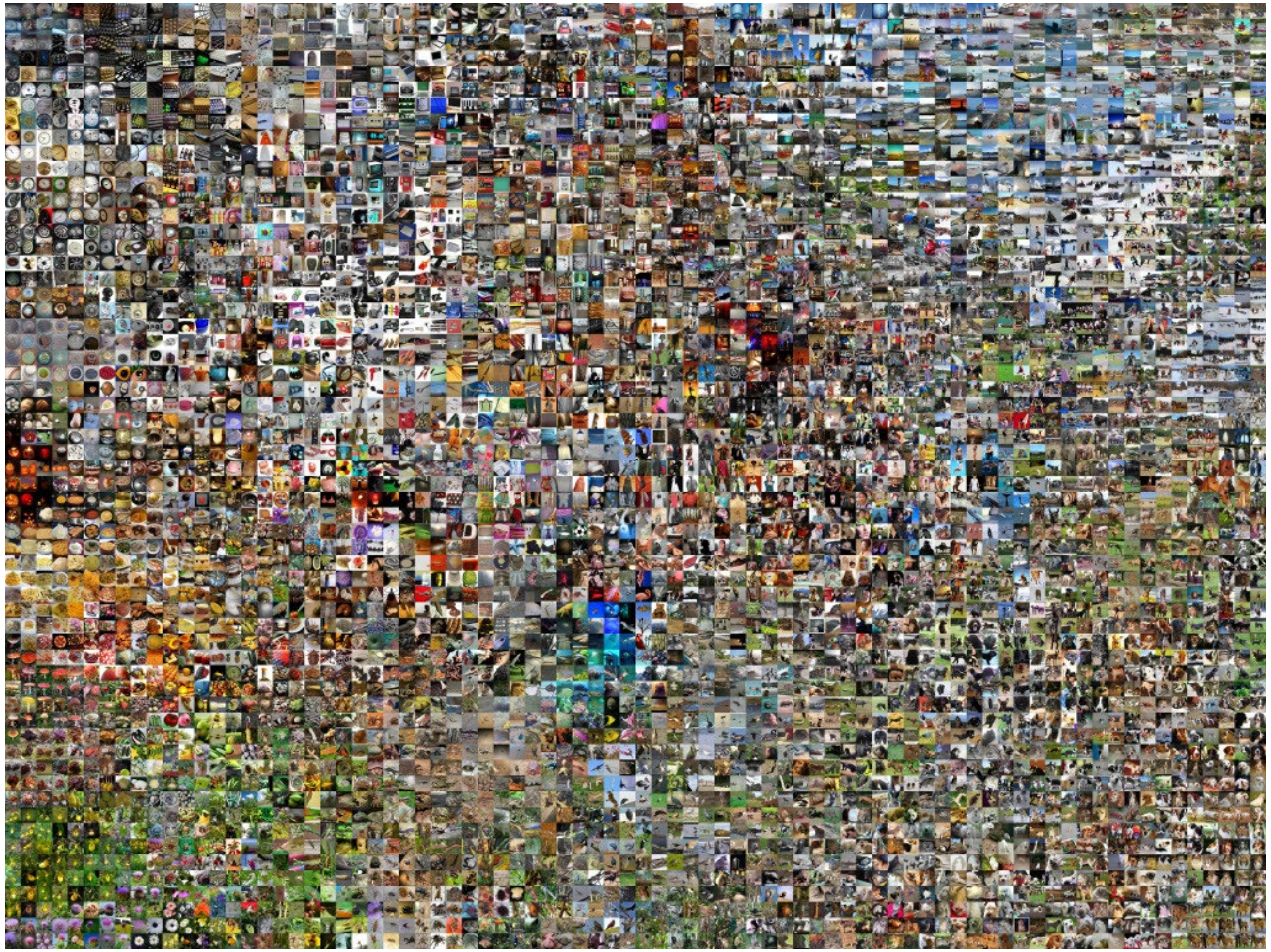




Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton. "Imagenet classification with deep convolutional neural networks." *Advances in neural information processing systems* 25 (2012).

How was this possible?

Large Collections of Annotated Data



The ImageNet project is a large visual database designed for use in visual object recognition software research. ***More than 14 million images*** *have been hand-annotated by the project to indicate what objects are pictured and in at least one million of the images, bounding boxes are also provided.[3]* ***ImageNet contains more than 20,000 categories***

From Wikipedia October 2021

J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li and L. Fei-Fei, ImageNet: A Large-Scale Hierarchical Image Database. *CVPR, 2009.*

Parallel Computing Architectures

And more recently…. Software libraries

TensorFlow

Google LLC, Public domain, via Wikimedia Commons

PyTorch

PyTorch, BSD <http://opensource.org/licenses/bsd-license.php>, via Wikimedia Commons

Data Scarcity

Training a CNN with Limited Aumont of Data

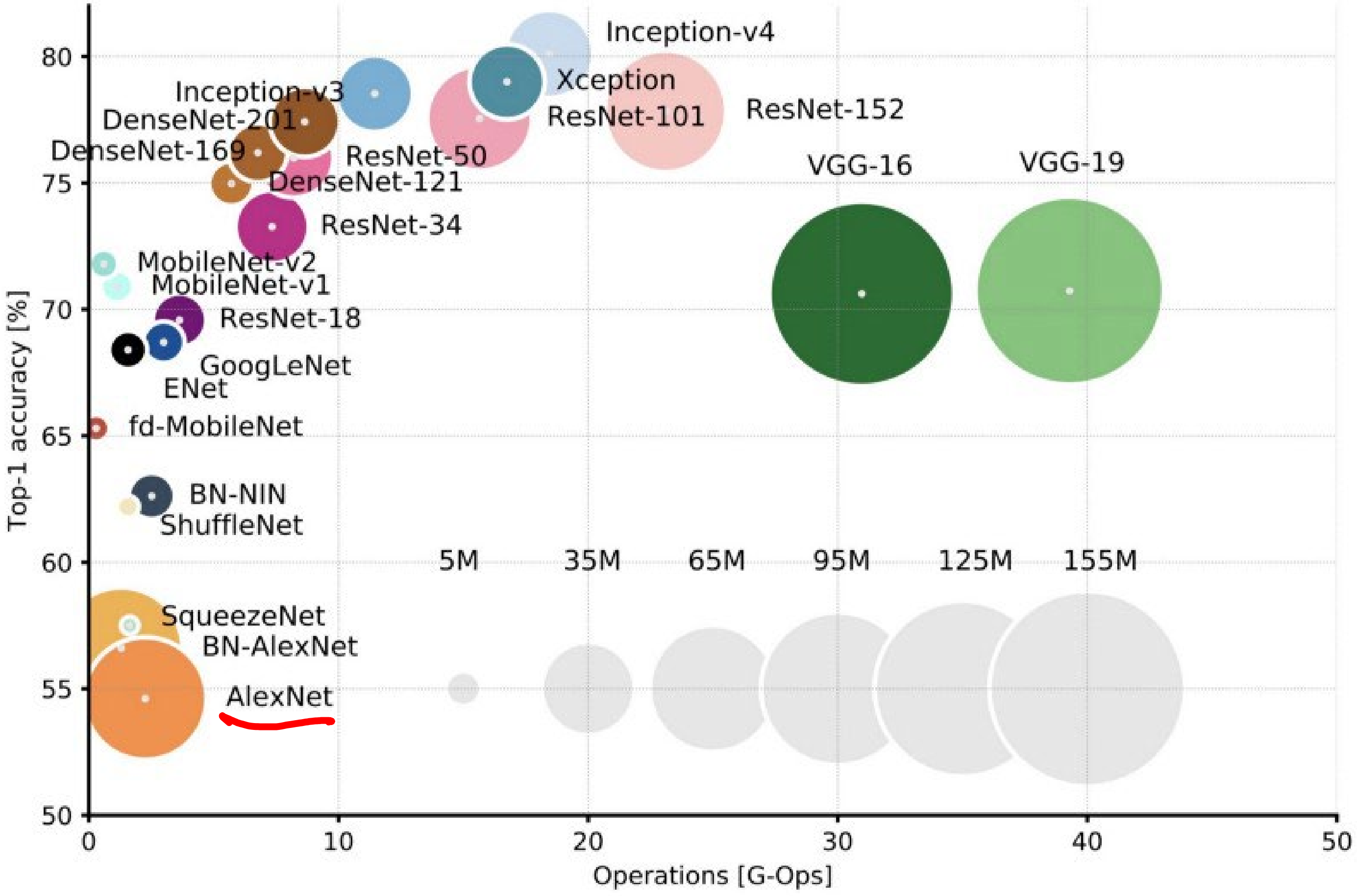
The need of data

Deep learning models are very data hungry.

Networks such as AlexNet have been trained on ImageNet datasets containing tens of thousands of images over hundreds of classes

The need of data

This is necessary to define millions of parameters characterizing these networks



Canziani, Alfredo, Adam Paszke, and Eugenio Culurciello. "An analysis of deep neural network models for practical applications." *arXiv preprint arXiv:1605.07678* (2016).

The need of data

Deep learning models are very data hungry.

... watch out: each image in the training set have to be annotated!

How to train a deep learning model with a few training images?

- Data augmentation
- Transfer Learning

Limited Amount of Data: Data Augmentation

Training a CNN with Limited Aumont of Data


Aleutian Islands
Stati Uniti
Messico
Steller sea lions in the western Aleutian Islands have declined 94% in the last 30 years.
3D

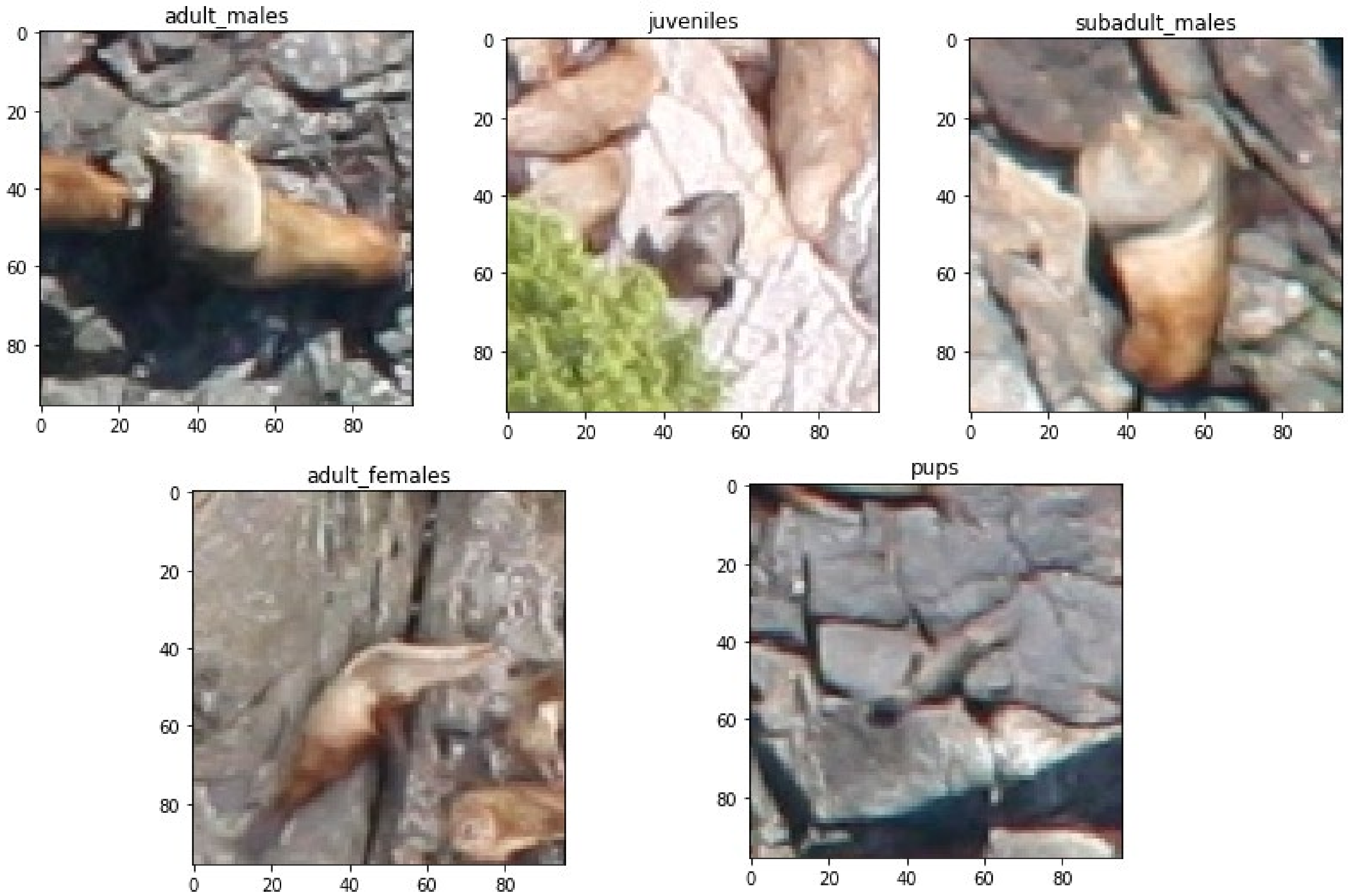
Kaggle in 2017 have opened a competition to develop algorithms which accurately count the number of sea lions in aerial photographs

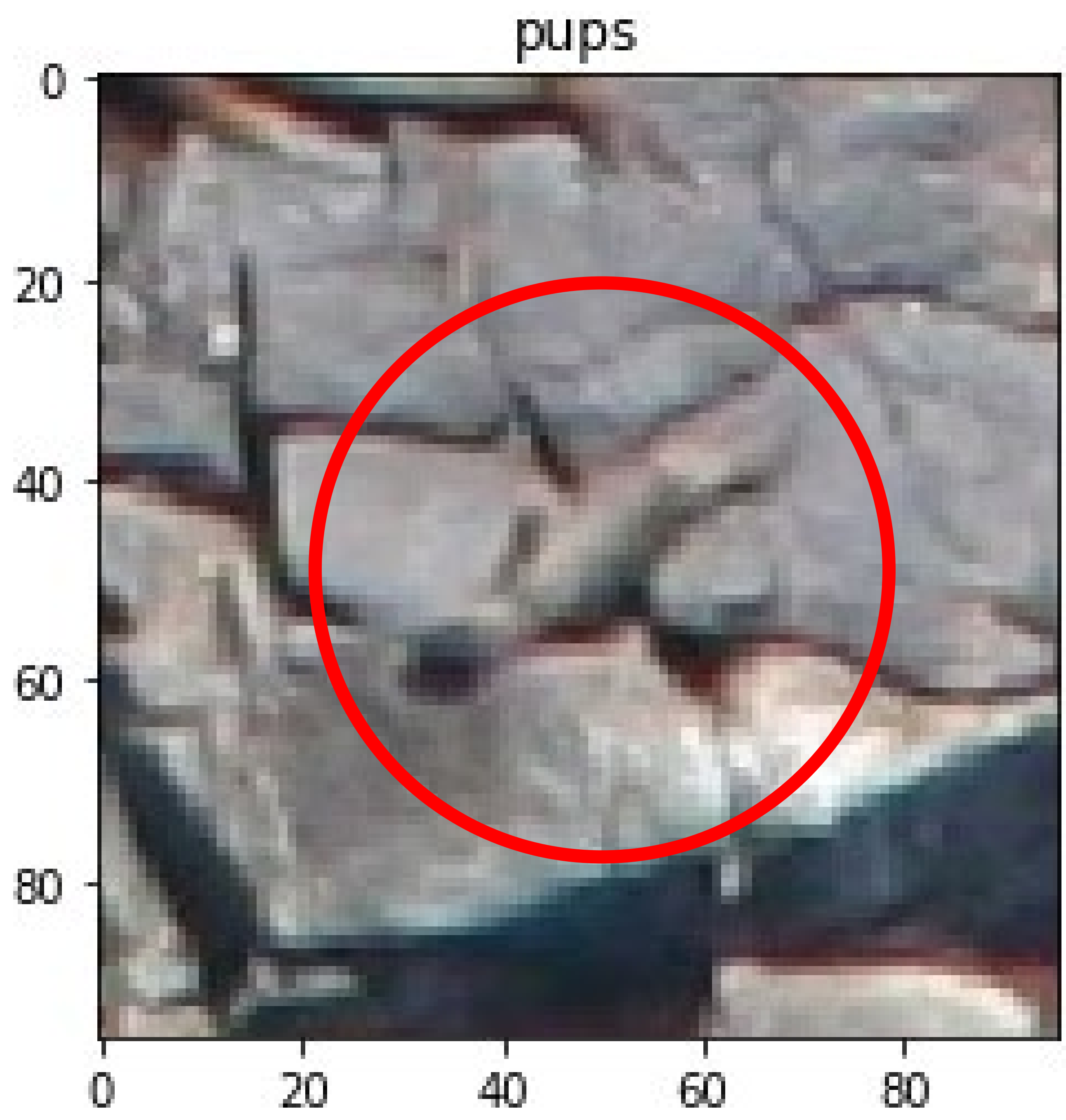
Credits Yinan Zhou

https://github.com/marioZYN/FC-CNN-Demo

The Challenge

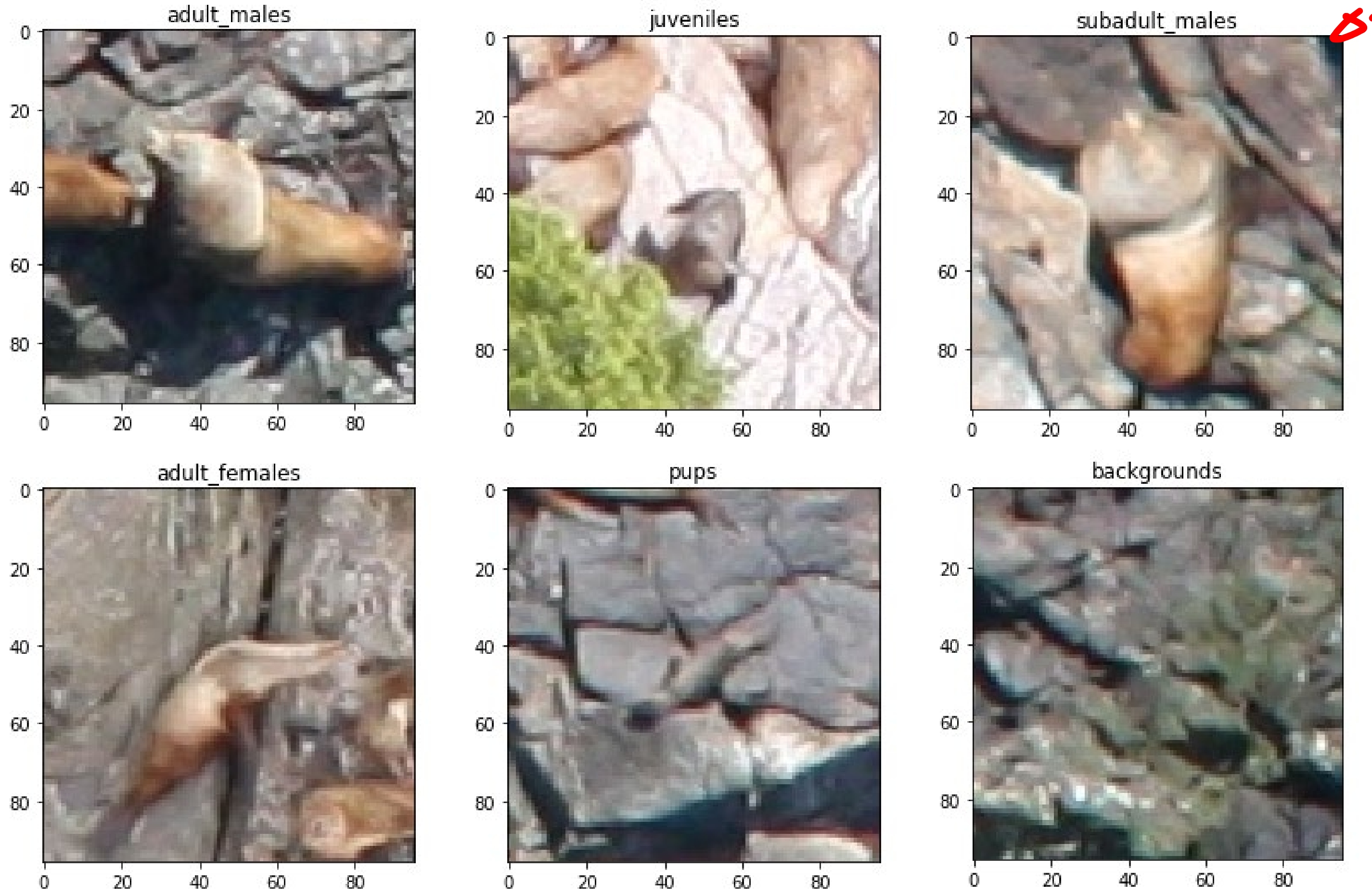
In very large aerial images (≈ 5K x 4K) shot by drones, automatically count the number of sealions per each category



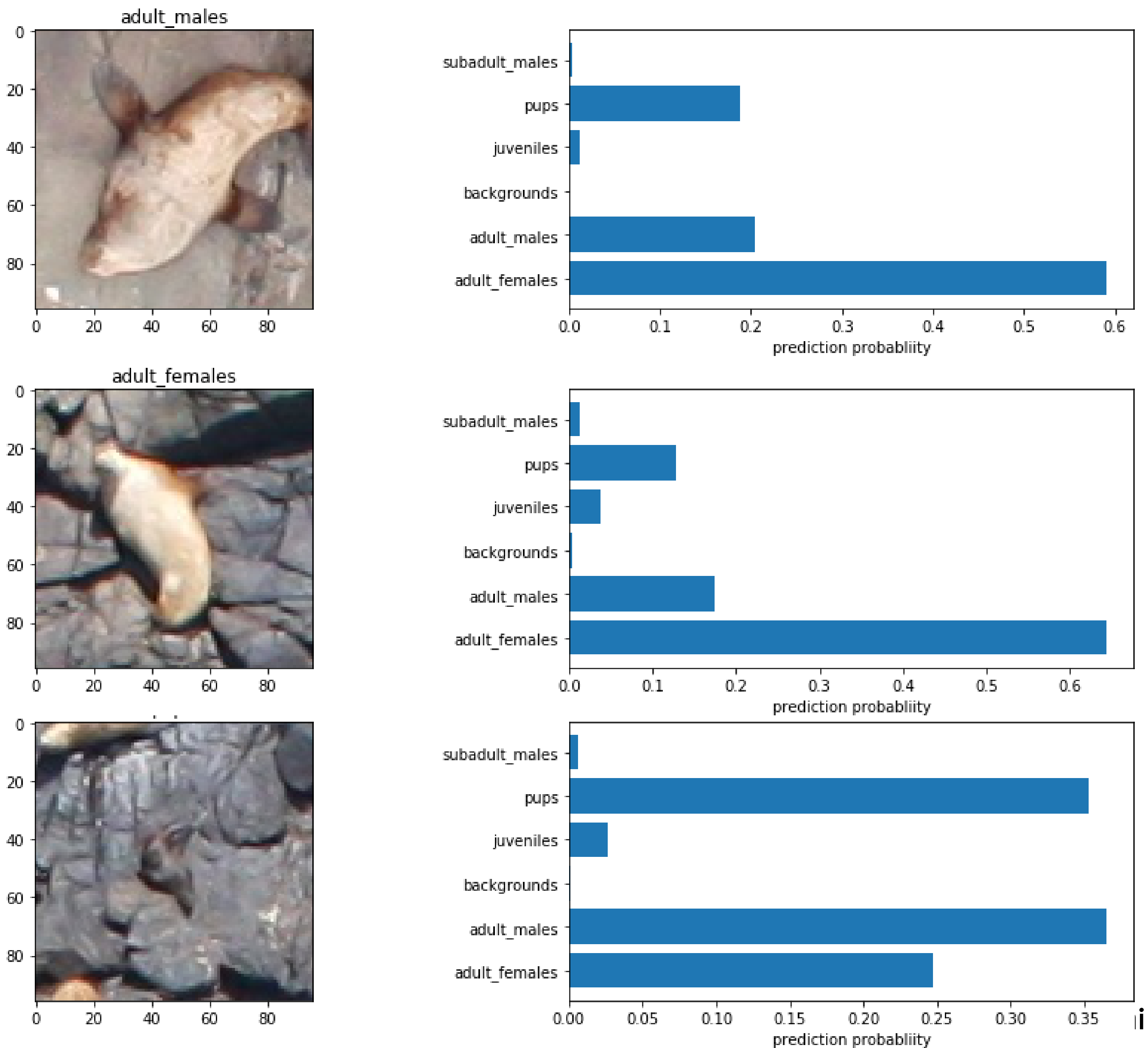

pups
0
20
40
60
80
0
20
40
60
80

The Challenge

This problem can be naively casted in a patch-by-patch 6-class classification problem, where we include also background



An Example of CNN predictions

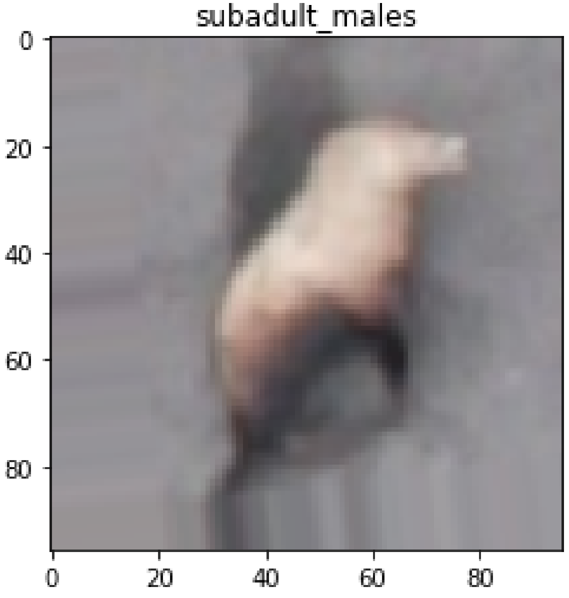


Credits Yinan Zhou
https://github.com/marioZYN/FC-CNN-Demo

Data Augmentation

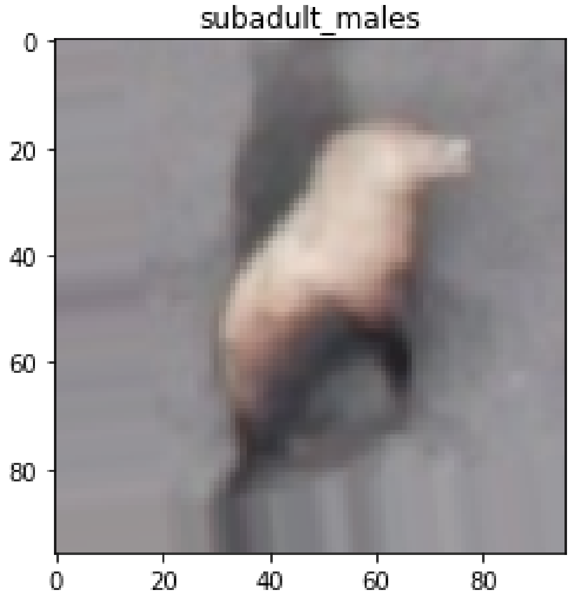
Often, each annotated image represents a class of images that are all likely to belong to the same class

In aereal photograps, for instance, it is normal to have rotated, shifted or scaled images without changing the label

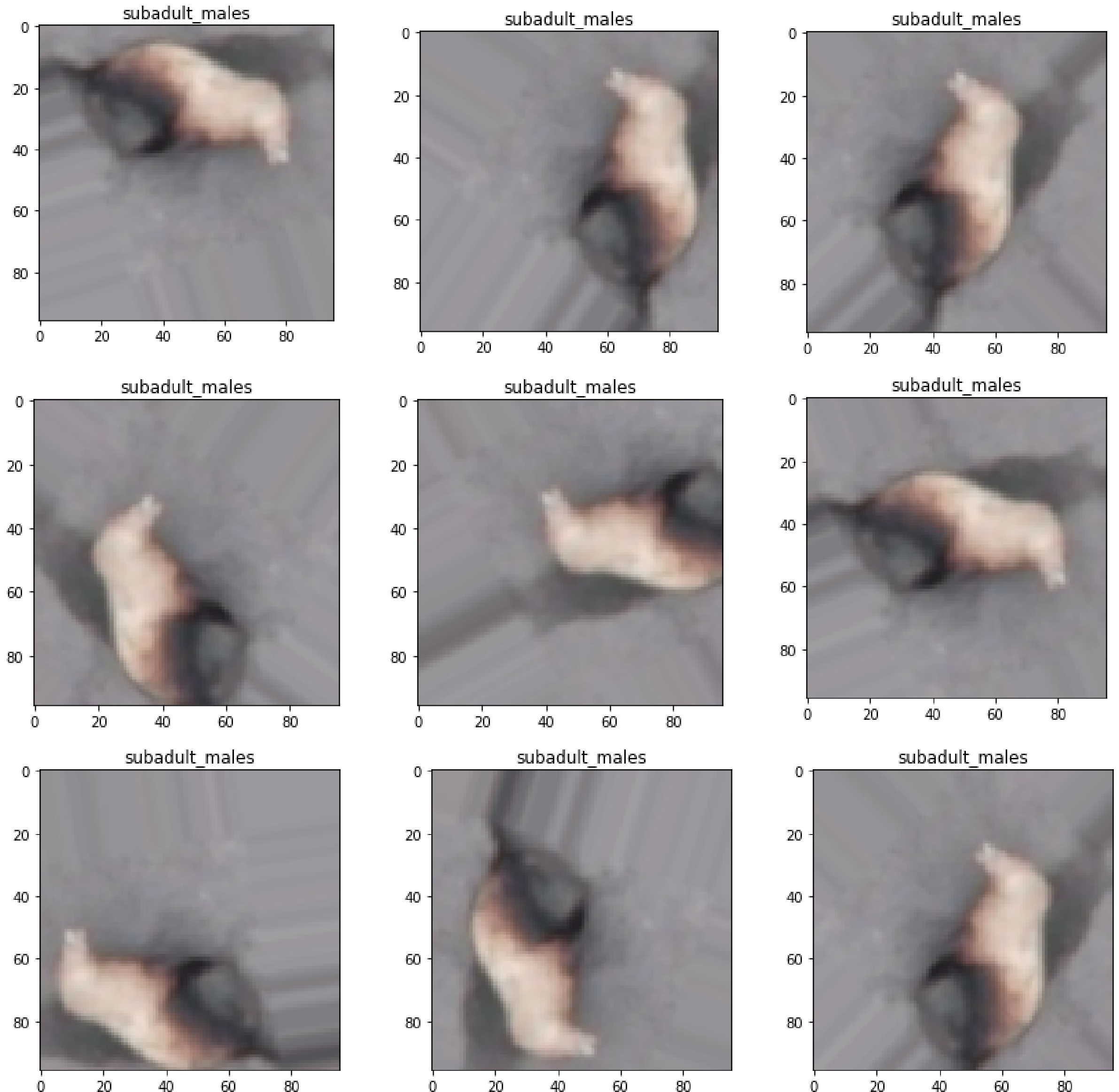


Data Augmentation

Original image



Augmented Images



Data Augmentation

Data augmentation is typically performed by means of

Geometric Transformations:

- Shifts /Rotation/Affine/perspective distortions
- Shear
- Scaling
- Flip

Photometric Transformations:

- Adding noise
- Modifying average intensity
- Superimposing other images
- Modifying image contrast

Data Augmentation Criteria

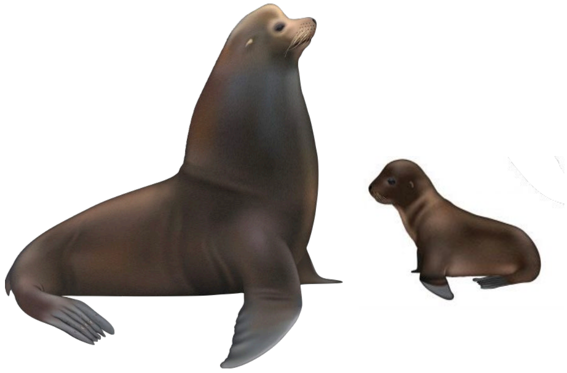
Data Augmentation: Criteria

Augmented versions **should preserve the input label**

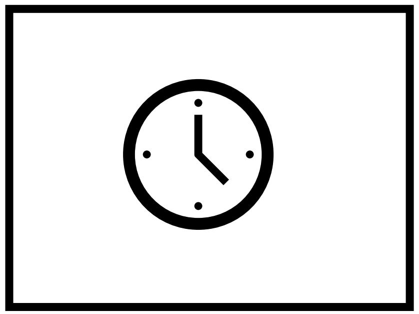
- e.g. if size/orientation is a key information to determine the output target (either the class or the value in case of regression), wisely consider scaling/rotation as transformation

Augmentation is meant to **promote network invariance** w.r.t. transformation used for augmentation

Non-preserving label augmentation

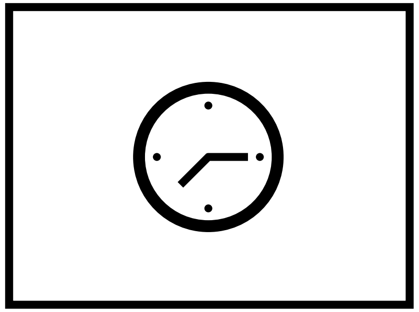


You don't want to introduce transformations that ruin distinctive information of a given class



16:00

(I, y)



16:00

$(R(I), y)$

A network predicting the time from an image of a clock without numbers is not invariant w.r.t rotations

Mixup Augmentation

Augmented copies $\{A_l(I)\}_l$ of an image I live in a ***vicinity*** of I, and **have the same label** of I

Transformations (photometric or geometric) are *expert-driven*

Mixup is a **domain-agnostic** data augmentation technique

- No need to know which (label-preserving) transformations to use
- mixup trains a neural network on *virtual samples* that are **convex combinations of pairs of examples and their labels**

Zhang, H., Cisse, M., Dauphin, Y. N., & Lopez-Paz, D. (2018). Mixup: Beyond empirical risk minimization. ICLR 2018

Mixup Augmentation

Given a pair of training samples (I_i, y_i) and (I_j, y_j) of drawn at random possibly belonging to different classes, we define

Virtual samples (and their label)

$$\tilde{I} = \lambda I_i + (1-\lambda) I_j$$
$$\tilde{y} = \lambda y_i + (1-\lambda) y_j$$

Where $\lambda \in [0,1]$ and y_i, and y_j are **one-hot encoded labels**

https://www.kaggle.com/code/kaushal2896/data-augmentation-tutorial-basic-cutout-mixup

Mixup Augmentation, Intuition

Mixup extends the training distribution by incorporating the prior knowledge that linear interpolations of feature vectors should lead to linear interpolations of the associated targets.

Mixup can be implemented in a few lines of codes and introduces minimal computation overhead.

Mixup in keras:
https://keras.io/guides/keras_cv/cut_mix_mix_up_and_rand_augment/

Zhang, H., Cisse, M., Dauphin, Y. N., & Lopez-Paz, D. (2018). Mixup: Beyond empirical risk minimization. ICLR 2018

The Benefits of Data Augmentation

Image Augmentation and CNN invariance

Given an annotated image (I, y) and a set of augmentation transformations $\{A_l\}_l$, we train the network using these pairs

$$\{(A_l(I), y)_l\}_l$$

Through data augmentation we train the network to «become invariant» to selected transformations. Since the same label is associated to I and $A_l(I)\ \forall l$

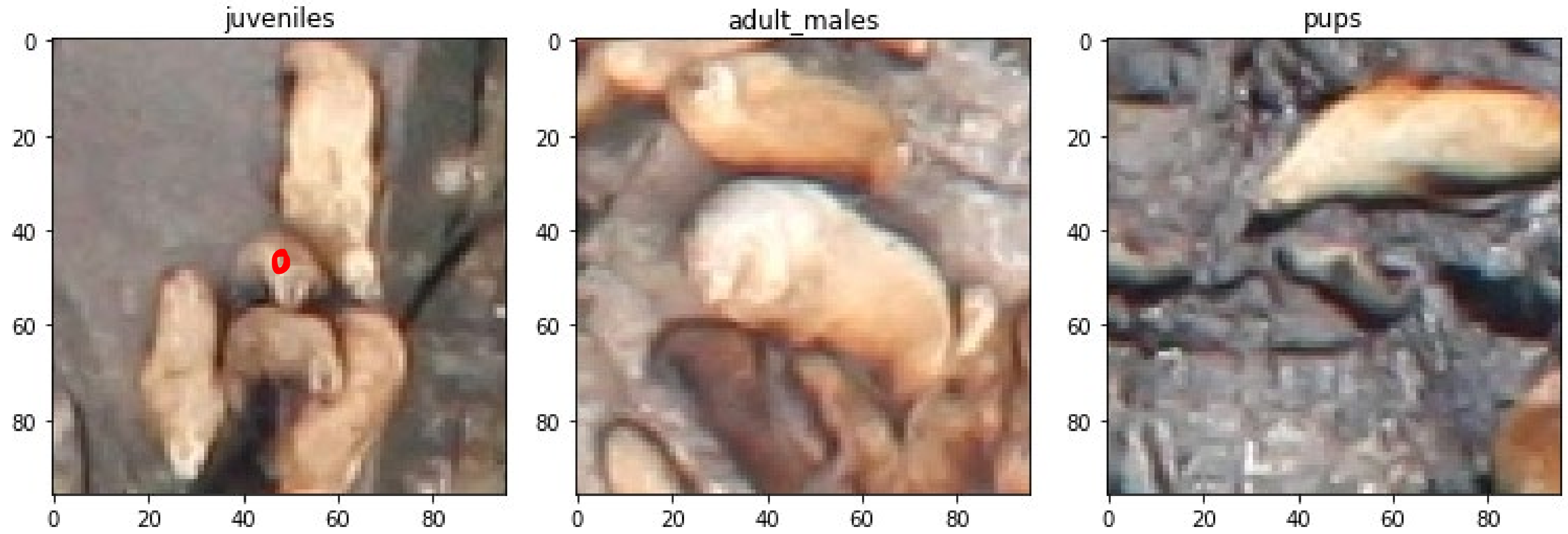
Unfortunately:

- invariance might not be always achieved in practice,
- several type of invariance cannot be achieved by synthetic manipulation of images

However…

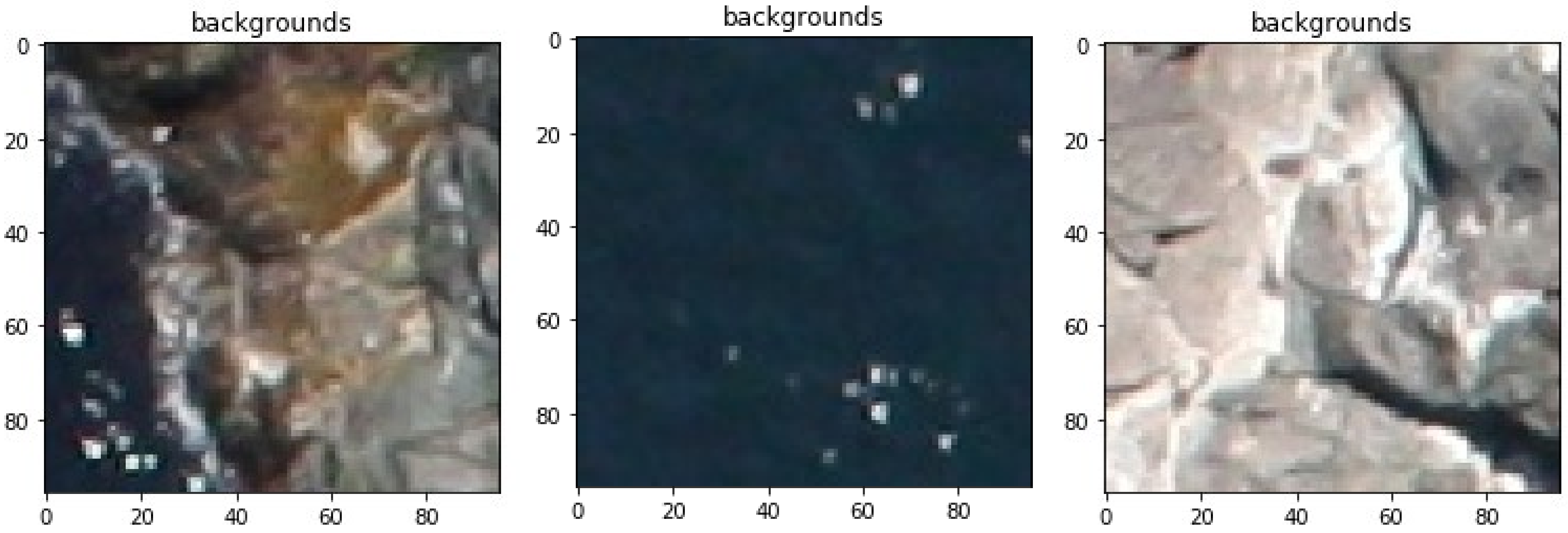
This sort of data augmentation might not be enough to capture the inter-class variability of images…

Superimposition of targets



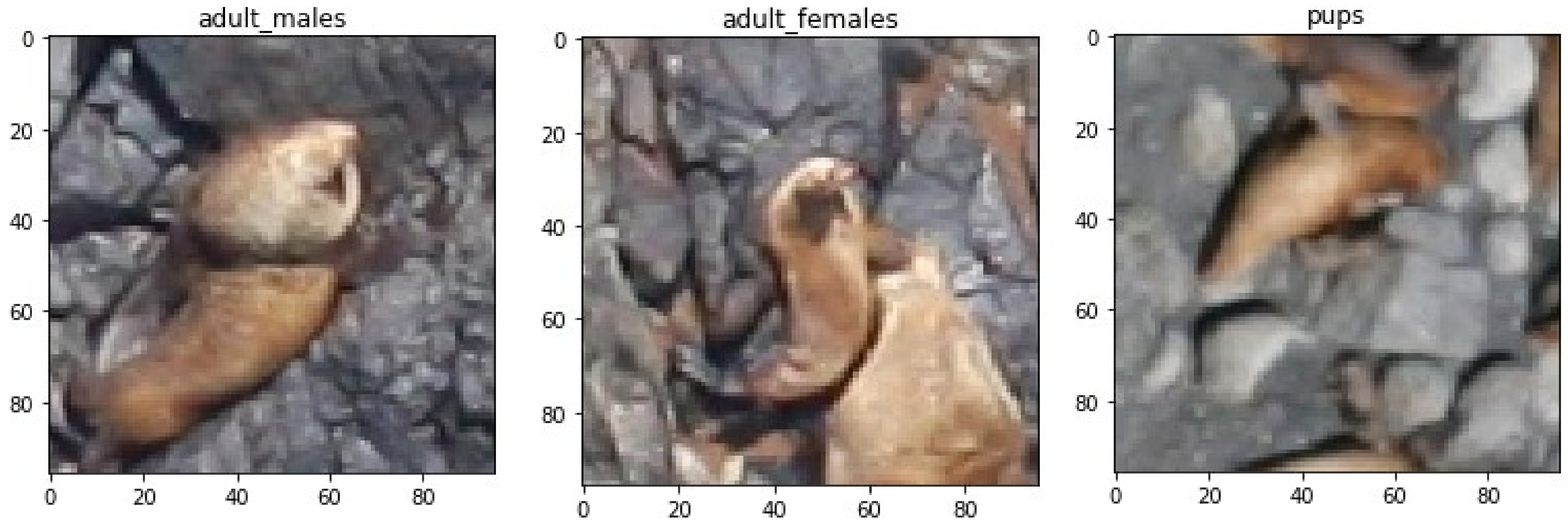
However…

Background variations



However…

Background variations



However…

Out of focus, bad exposure

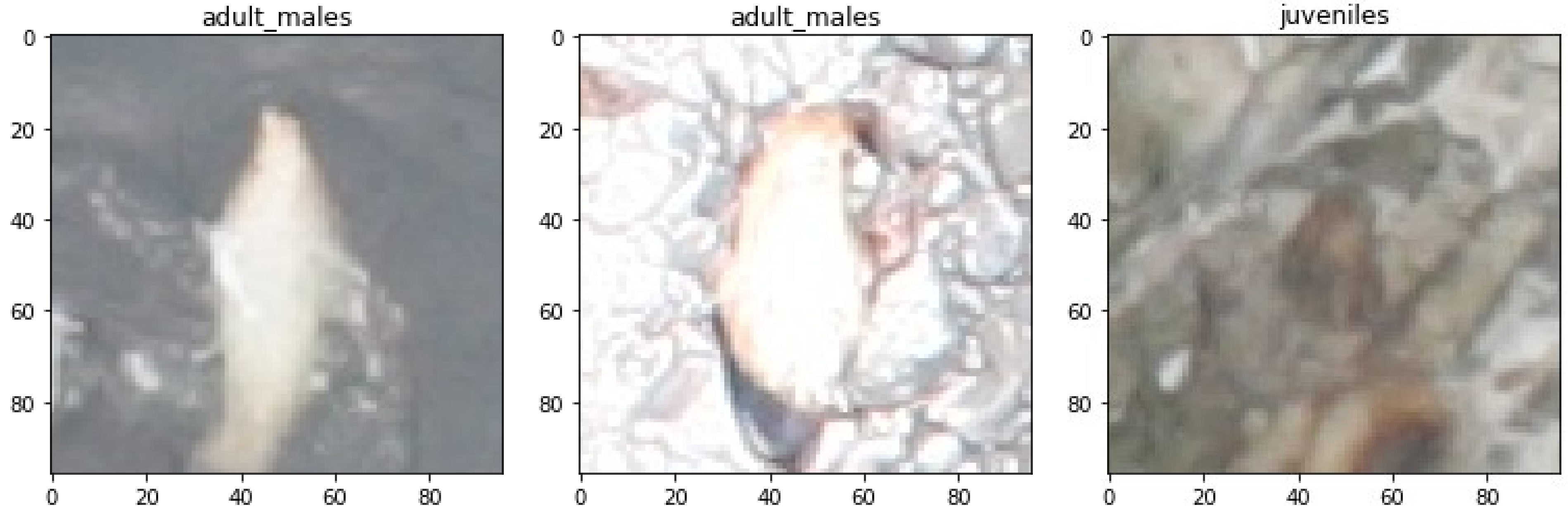


Image Augmentation and Overfitting

Given an annotated image (I, y) and a set of augmentation transformations $\{A_l\}_l$, we train the network using these pairs

$$\{(A_l(I), y)_l\}_l$$

Training including augmentation **reduces the risk of overfitting**, as it significantly increase the training set size.

Note: data augmentation can be implemented as a network layer, such that it is executed on each batch, **thus changing augmented images at each epochs**

Image Augmentation and Class Imbalance

Moreover, data augmentation can be used to compensate for class imbalance in the training set, by **creating more realistic examples from the minority class**

In general, **transformations used in data-augmentation** $\{A_l\}$ **can be also class-specific**, in order to preserve the image label

Watch out

If Data-augmentation introduces some «hidden traces» that are class-discriminative, then the network will learn these to perform detection!

For instance

- Blurring only images of a specific class, makes the network learn that class *“as blurry”*, despite the image semantics. This holds for interpolation artifacts as well
- Changing colors / creating inconsistencies / introducing minor padding artifacts in a certain class of images, might create new class-discriminative patterns.

Test Time Augmentation

Test Time Augmentation (TTA) or Self-ensembling

Even if the CNN is trained using augmentation, **it won't achieve perfect invariance** w.r.t. considered transformations

Test time augmentation (TTA): augmentation can be also performed at test time to improve prediction accuracy.

- Perform a few random augmentation of each test image I
$$\{A_l(I)\}_l$$
- Classify all the augmented images and save the posterior vectors
$$\boldsymbol{p_l} = CNN(A_l(I))$$
- Define the CNN prediction by aggregating the posterior vectors $\{\boldsymbol{p}_l\}$
$$\text{e.g.}\ \boldsymbol{p} = Avg(\{\boldsymbol{p}_l\}_l)$$

$I \Rightarrow \boxed{\text{CNN}} \Rightarrow \hat{y}$

$I \Rightarrow \boxed{\text{AUG}} \Rightarrow \{A_t(I)\} \Rightarrow \boxed{\text{CNN}} \Rightarrow \{\hat{y}_t\} \Rightarrow \boxed{\text{aggregation}} \Rightarrow \hat{y}$

Test Time Augmentation (TTA) or Self-ensembling

TTA:

- particularly useful for test images where the model is quite unsure.
- extremely computationally demanding

Need to wisely configure the number and type of transformations to be performed at test time

Test Time Augmentation

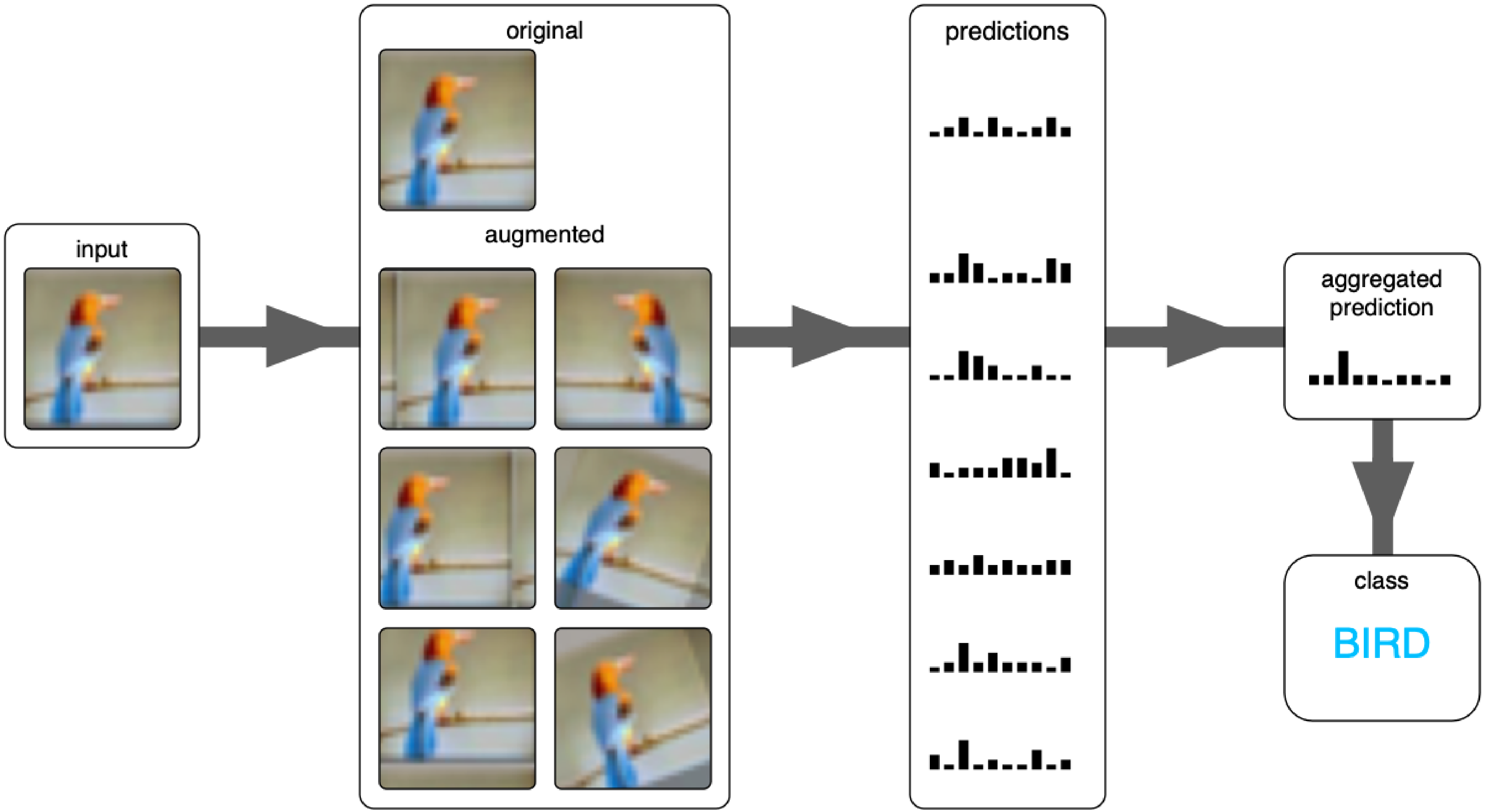


Figure source: https://stepup.ai/test_time_data_augmentation/

Augmentation In Keras

Augmentation in Keras

There are multiple **preprocessing layers** to be introduced after the input layer to perform:

- photometric transformations
- geometric transformations

to the image

https://keras.io/api/layers/preprocessing_layers/image_augmentation/

Augmentation Layers

These layers apply random augmentation transforms to a batch of images. **They are only active during training.**

tf.keras.layers.RandomCrop

tf.keras.layers.RandomFlip

tf.keras.layers.RandomTranslation

tf.keras.layers.RandomRotation

tf.keras.layers.RandomZoom

tf.keras.layers.RandomHeight

tf.keras.layers.RandomWidth

tf.keras.layers.RandomContrast

Preprocessing Layers

Image preprocessing layers, these are active at inference

- [Resizing layer]
- [Rescaling layer]
- [CenterCrop layer]

Augmenting Images

Define a simple network that performs a random flip of the input

```
flip = tf.keras.Sequential([
  tfkl.RandomFlip("horizontal_and_vertical"),
])
```

Invoke this network to apply augmentation to images

```
flipped_X_train = flip(X_train)
```

Augmenting Images

You can stuck multiple layers

```
# pack a few augmentation layers in a sequence
augmentationNet = tf.keras.Sequential([
    tfkl.RandomFlip("horizontal_and_vertical"),
    tfkl.RandomTranslation(0.1,0.1),
    tfkl.RandomRotation(0.1),
], name='augmentationNet')
```

Invoke this network to apply augmentation to images

```
augmentated_X_train = augmentationNet(X_train)
```

Training with data augmentation

You can include augmentation / preprocessing layers directly in the network architecture

Note:

- Augmentation layers will be active only during training
- Preprocessing layers will be active also during inference

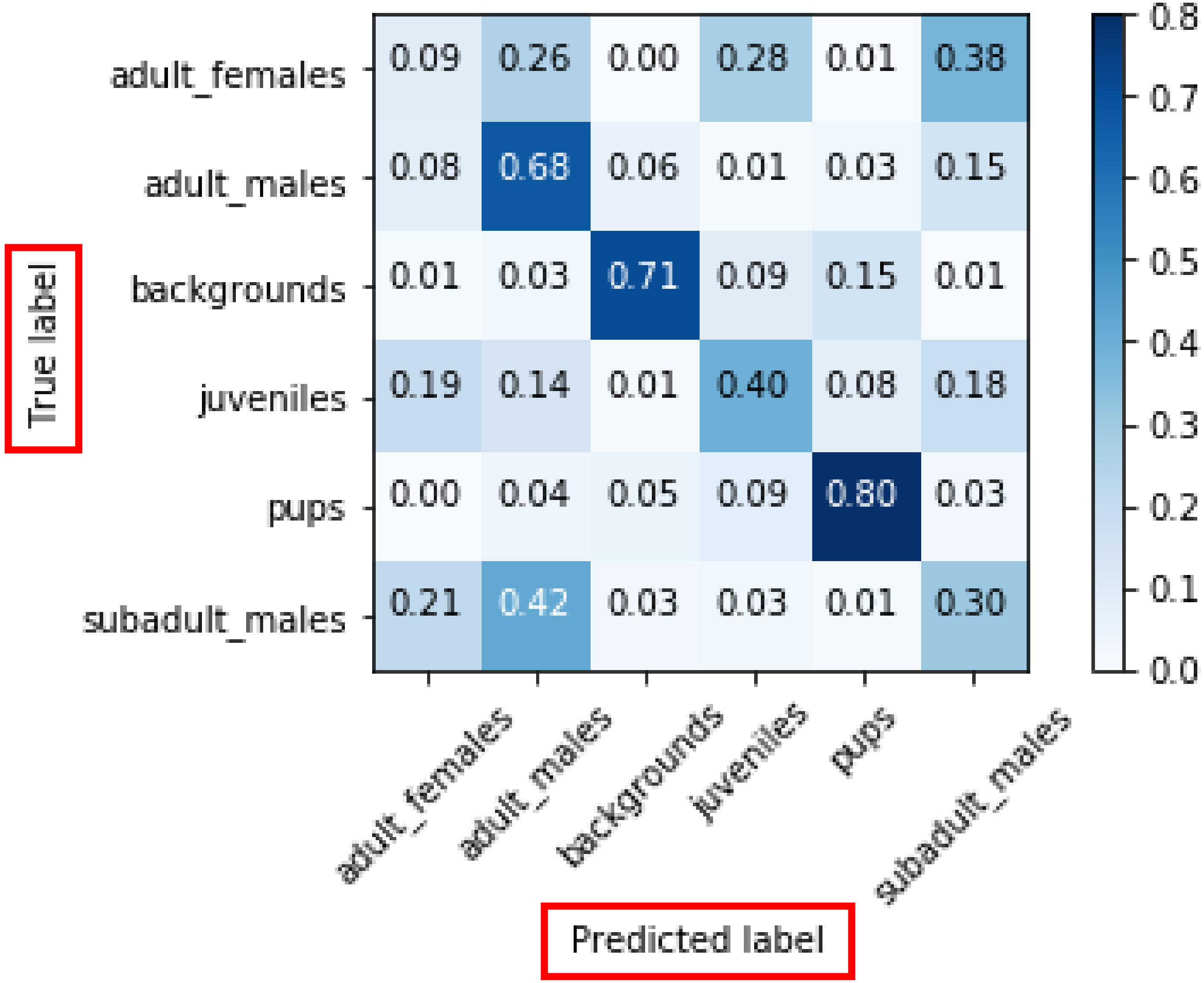
```
def build_model_with_augmentation(input_shape, output_shape):
    tf.random.set_seed(seed)

    # Build the neural network layer by layer
    input_layer = tfkl.Input(shape=input_shape, name='Input')

    # include augmentation layers
    a = tfkl.RandomFlip("horizontal_and_vertical")(input_layer)
    b = tfkl.RandomTranslation(0.1,0.1)(a)
    c = tfkl.RandomRotation(0.1)(b)

    conv1 = tfkl.Conv2D(…)(c)
```

A bit more of background

Performance measures

and an overview of successful architectures

Confusion Matrix

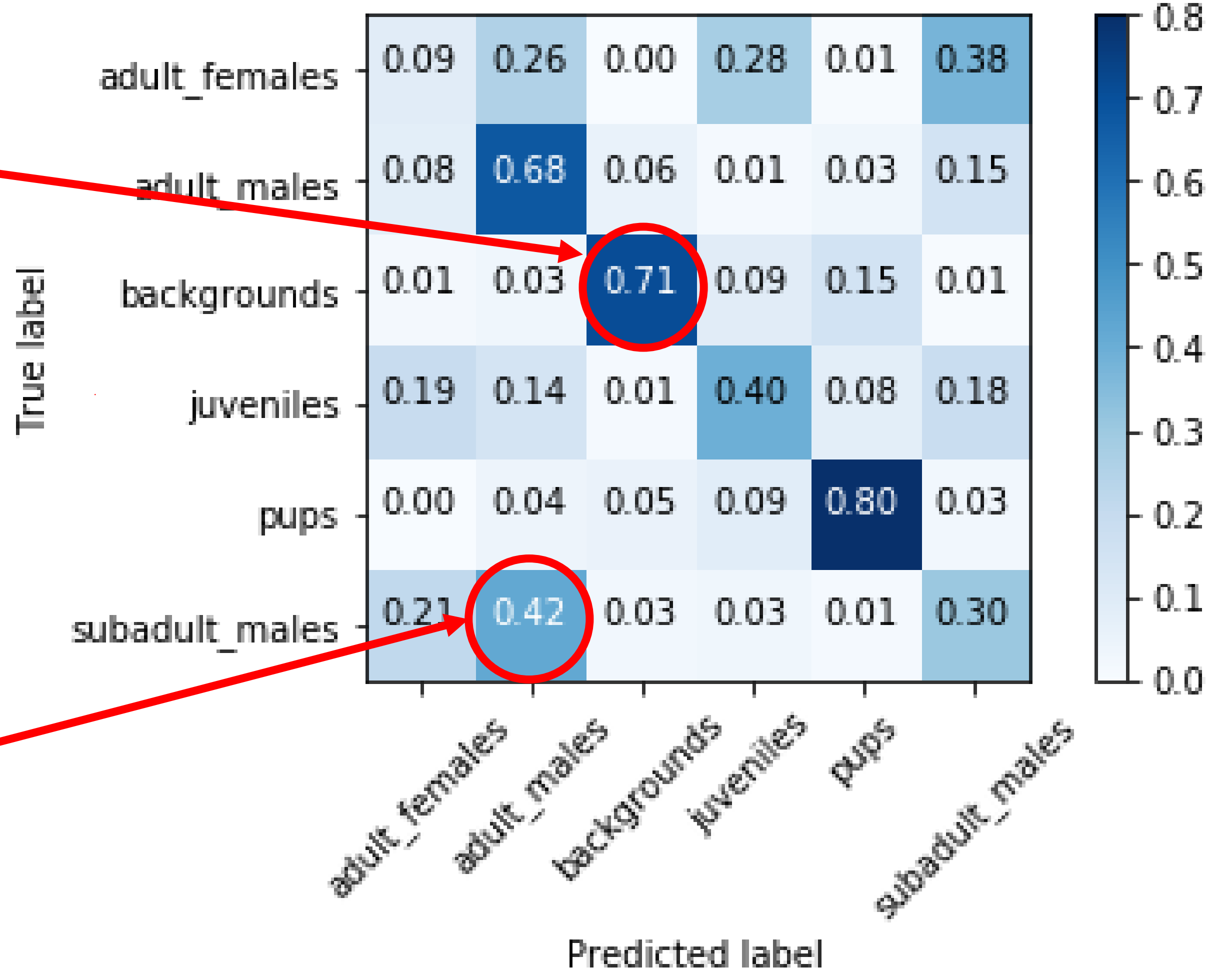


Confusion Matrix

The element $C(i,j)$ i.e. at the i-th row and j-th column corresponds to the percentage of elements belonging to class i classified as elements of class j

71% of background patches have been correctly classified as background

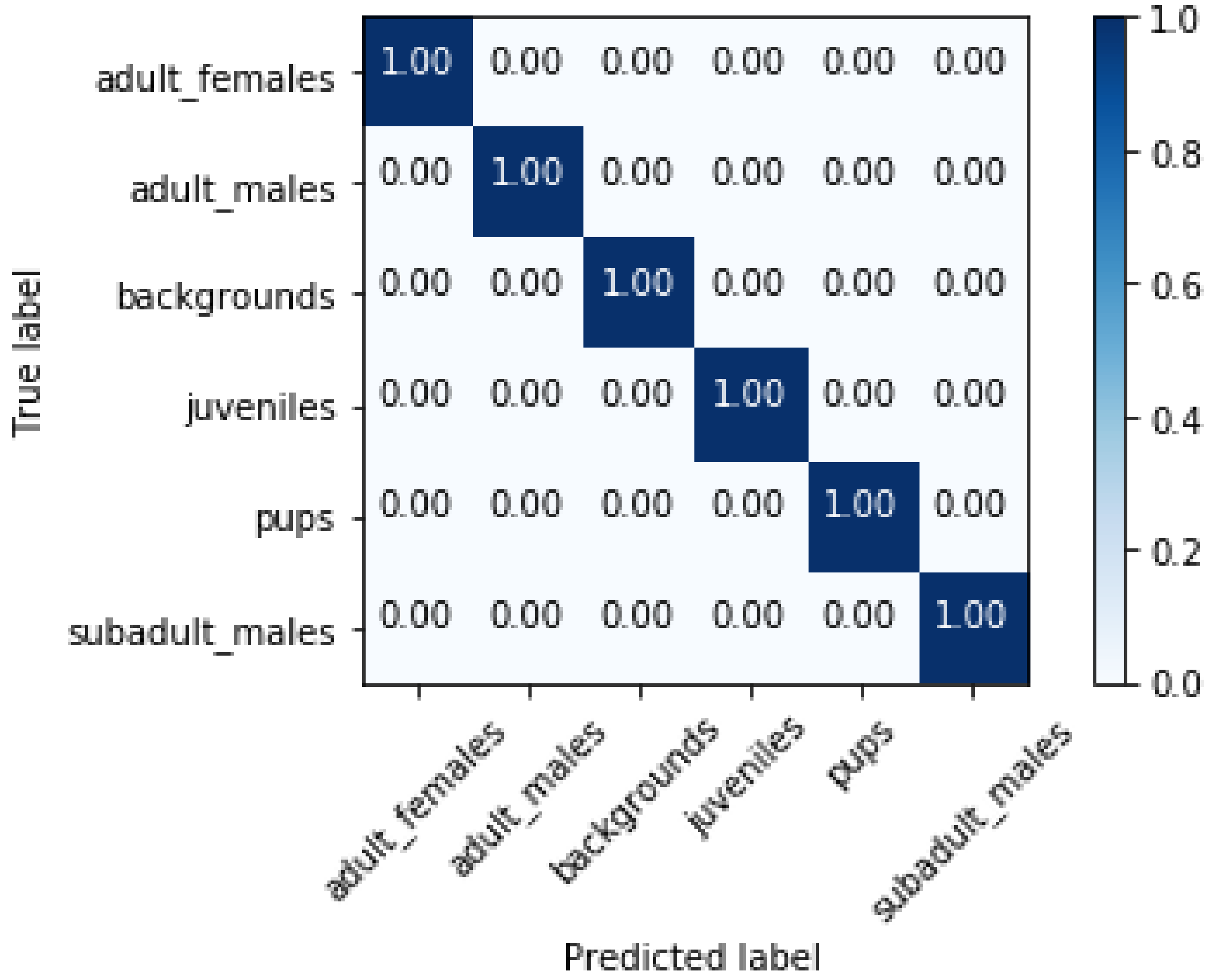
42% of sub-adult males patches have been wrongly classified as adult-males



hi

… so, the ideal confusion matrix

Which rarely happens



Two-Class Classification

Background:

In a two-class classification problem (binary classification), the **CNN output is equivalent to a scalar,** since

$$CNN(I) = [p, 1 - p]$$

being p the probability of I to belong to the first class.

Thus, we can write

$$CNN(I) = p$$

Then, we can decide that I belongs to the first class when

$$CNN(I) > \Gamma$$

and use Γ different from 0.5, which is the standard.

We require stronger evidence before claiming I **belongs to class 1.**

Changing Γ **establishes a trade off between FPR and TPR.**

Two-Class Classification

Classification performance in case of **binary classifiers** can be also measured in terms of the **ROC** (receiver operating characteristic) **curve**, which does not depend on the threshold you set for each class

This is useful in case you plan to modify this and not use $\mathbf{0.5}$

The ideal detector would achieve:

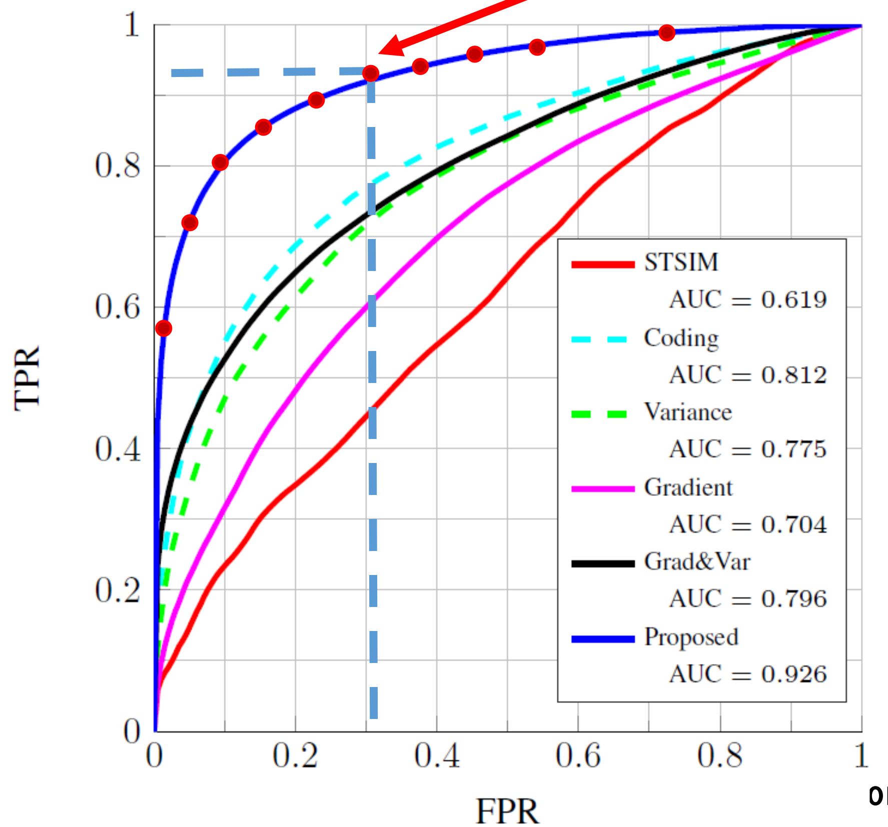
- $FPR = 0\%$,
- $TPR = 100\%$

Thus, the closer to (0,1) the better

The largest the **Area Under the Curve** (AUC), the better

The optimal parameter is the one yielding the point closest to (0,1)

(FPR, TPR) for a specific parameter



CNN for Quality Inspection

In collaboration with

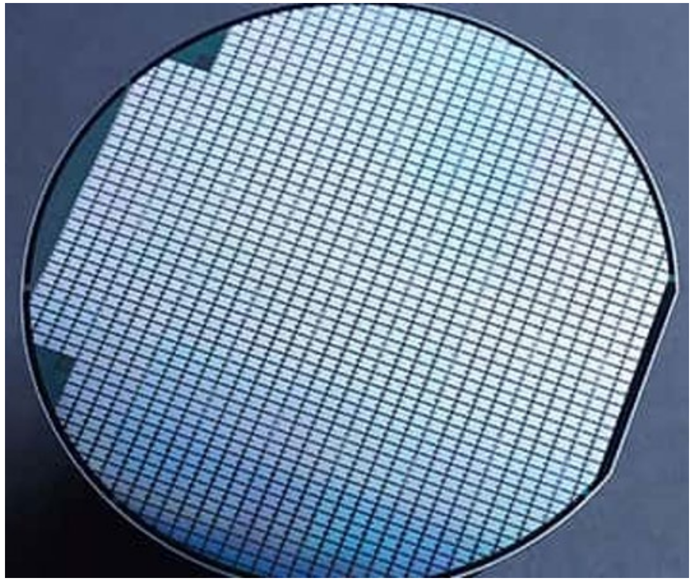
life.augmented

Giacomo Boracchi

Scenario

Chip Manufacturer

Silicon Wafer



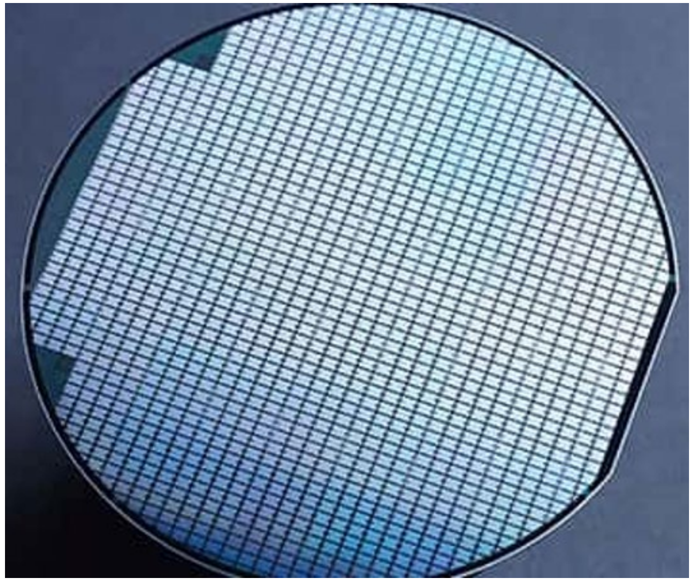
Chips / Memories / Sensors are everywere

In collaboration with

Frittoli, L., Carrera, D., Rossi, B., Fragneto, P., & Boracchi, G. (2022). Deep open-set recognition for silicon wafer production monitoring. *Pattern Recognition*, *124*, 108488.

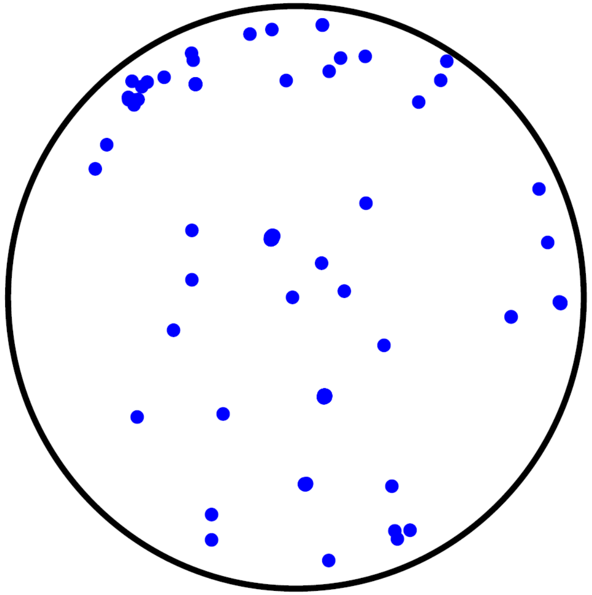
Monitoring Silicon Wafer Manufacturing Process

Wafer



Inspection Tool

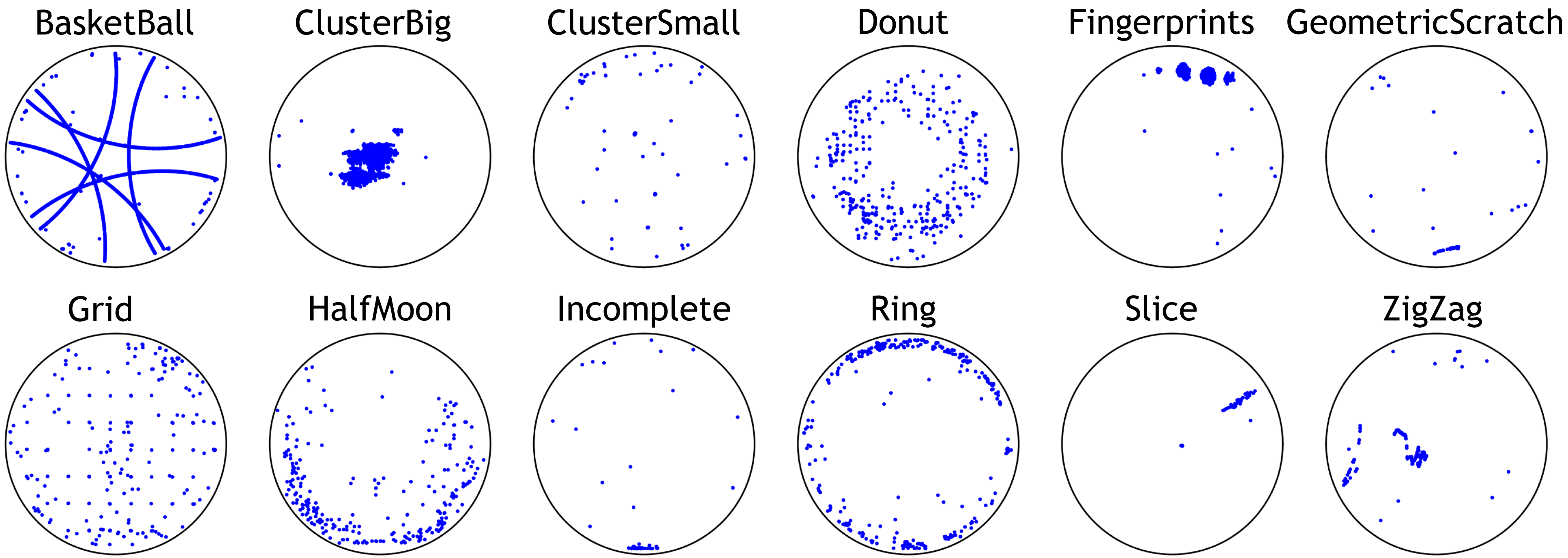
Wafer Defect Map (WDM)



Frittoli, L., Carrera, D., Rossi, B., Fragneto, P., & Boracchi, G. (2022). Deep open-set recognition for silicon wafer production monitoring. *Pattern Recognition*, *124*, 108488.

Classess of WDM Patterns

Specific **patterns** in WDMs might indicate **problems** in the production

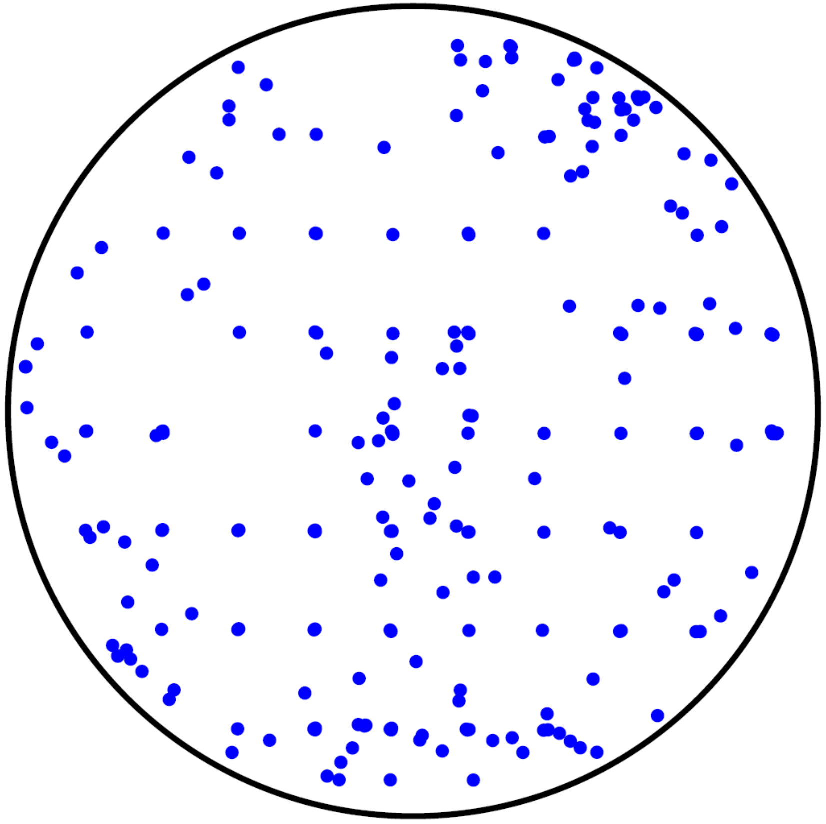


Classify WDM to raise prompt alerts

Frittoli, L., Carrera, D., Rossi, B., Fragneto, P., & Boracchi, G. (2022). Deep open-set recognition for silicon wafer production monitoring. *Pattern Recognition*, *124*, 108488.

Challenges

- Huge resolution: a WDM as a grayscale would require ~ 3 GB to store w in memory
- Very Limited Supervision
- Some defects occur very rarely

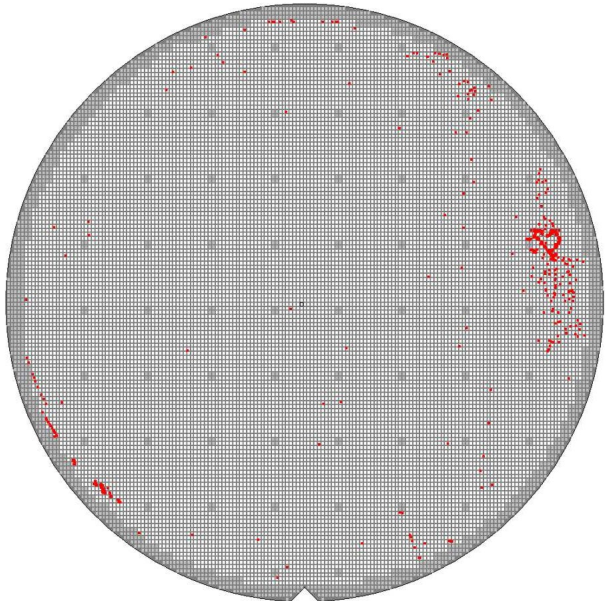


Frittoli, L., Carrera, D., Rossi, B., Fragneto, P., & Boracchi, G. (2022). Deep open-set recognition for silicon wafer production monitoring. *Pattern Recognition*, *124*, 108488.

Our CNN

Collaboration with

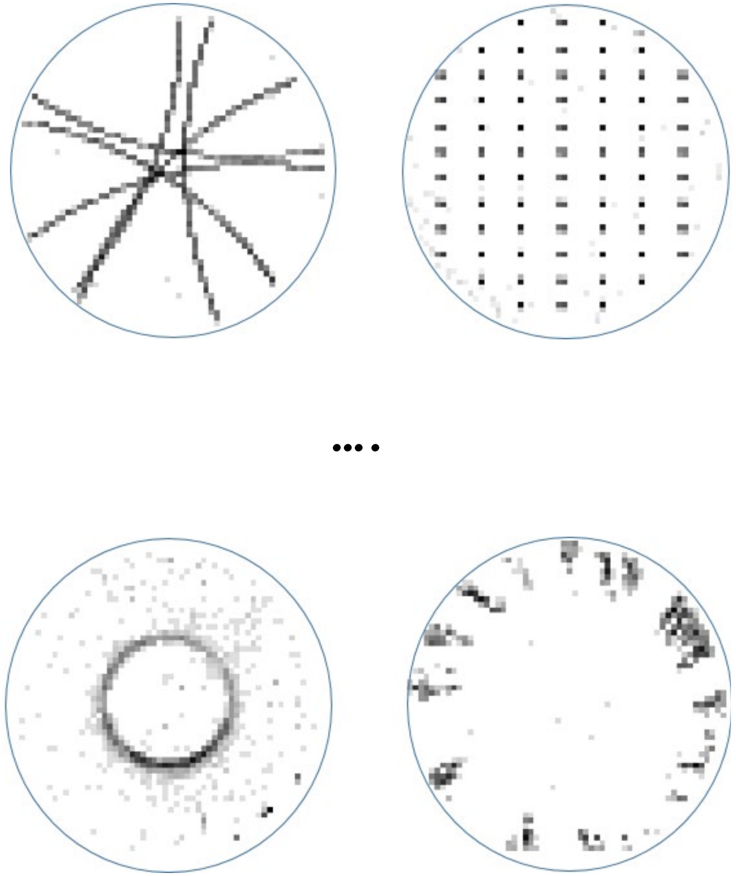
Train a deep learning model to identify defective patterns



Wafer Defect Map

Deep Learning

Defect Patterns



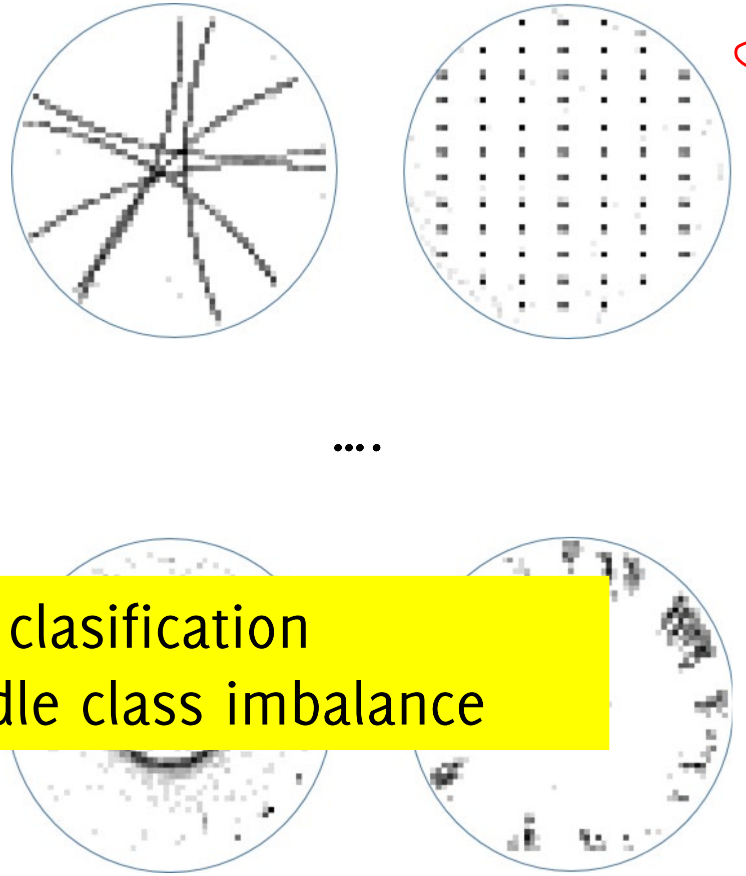
R. di Bella, D. Carrera, B. Rossi, P. Fragneto, G. Boracchi «Wafer Defect Map Classification Using Sparse Convolutional Networks” ICIAP 2019

Data Augmentation is often key..

Collaboration with ST life.augmented

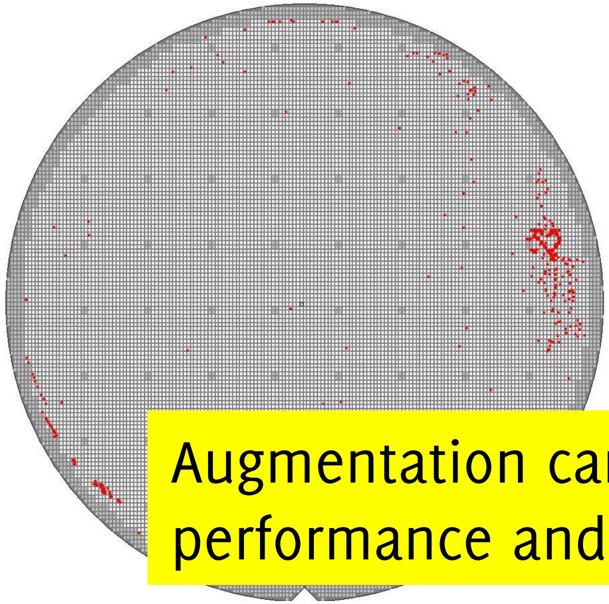
Train a deep learning model to identify defective patterns

Defect Patterns



....

Deep Learning

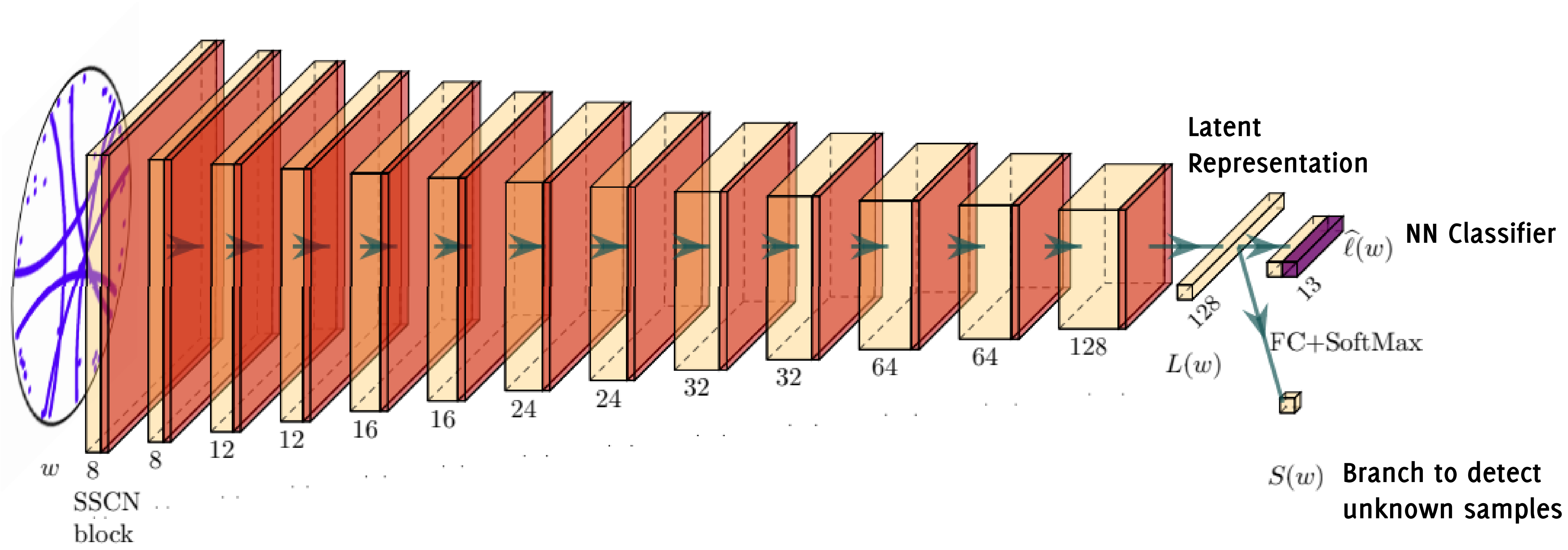


Wafer Defect Map

Augmentation can greatly improve clasification performance and successfully handle class imbalance

R. di Bella, D. Carrera, B. Rossi, P. Fragneto, G. Boracchi «Wafer Defect Map Classification Using Sparse Convolutional Networks” ICIAP 2019

Our CNN



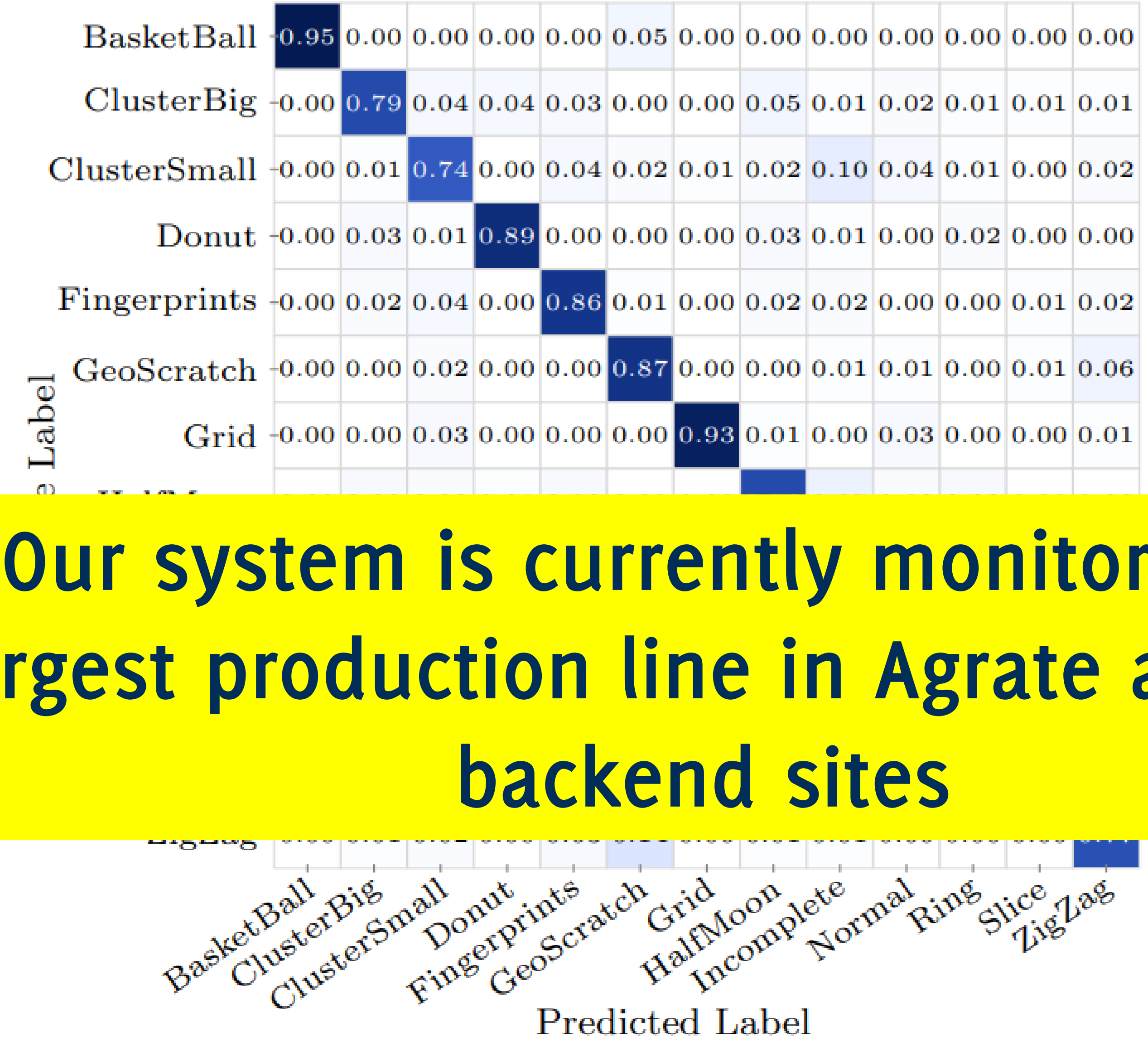
Frittoli, L., Carrera, D., Rossi, B., Fragneto, P., & Boracchi, G. (2022). Deep open-set recognition for silicon wafer production monitoring. *Pattern Recognition*, *124*, 108488.

Results

True Label \ Predicted Label	BasketBall	ClusterBig	ClusterSmall	Donut	Fingerprints	GeoScratch	Grid	HalfMoon	Incomplete	Normal	Ring	Slice	ZigZag
BasketBall	0.95	0.00	0.00	0.00	0.00	0.05	0.00	0.00	0.00	0.00	0.00	0.00	0.00
ClusterBig	0.00	0.79	0.04	0.04	0.03	0.00	0.00	0.05	0.01	0.02	0.01	0.01	0.01
ClusterSmall	0.00	0.01	0.74	0.00	0.04	0.02	0.01	0.02	0.10	0.04	0.01	0.00	0.02
Donut	0.00	0.03	0.01	0.89	0.00	0.00	0.00	0.03	0.01	0.00	0.02	0.00	0.00
Fingerprints	0.00	0.02	0.04	0.00	0.86	0.01	0.00	0.02	0.02	0.00	0.00	0.01	0.02
GeoScratch	0.00	0.00	0.02	0.00	0.00	0.87	0.00	0.00	0.01	0.01	0.00	0.01	0.06
Grid	0.00	0.00	0.03	0.00	0.00	0.00	0.93	0.01	0.00	0.03	0.00	0.00	0.01
HalfMoon	0.00	0.03	0.02	0.01	0.03	0.00	0.00	0.79	0.07	0.02	0.03	0.00	0.00
Incomplete	0.00	0.00	0.04	0.00	0.01	0.00	0.00	0.01	0.90	0.02	0.03	0.00	0.00
Normal	0.00	0.00	0.01	0.00	0.00	0.00	0.00	0.01	0.02	0.93	0.00	0.00	0.00
Ring	0.00	0.00	0.01	0.01	0.00	0.00	0.00	0.03	0.10	0.01	0.82	0.00	0.00
Slice	0.00	0.00	0.00	0.00	0.01	0.00	0.00	0.00	0.00	0.00	0.00	0.99	0.00
ZigZag	0.00	0.01	0.02	0.00	0.03	0.14	0.00	0.01	0.01	0.00	0.00	0.00	0.77

Frittoli, L., Carrera, D., Rossi, B., Fragneto, P., & Boracchi, G. (2022). Deep open-set recognition for silicon wafer production monitoring. *Pattern Recognition*, *124*, 108488.

Results



Our system is currently monitoring the largest production line in Agrate and most backend sites

Frittoli, L., Carrera, D., Rossi, B., Fragneto, P., & Boracchi, G. (2022). Deep open-set recognition for silicon wafer production monitoring. *Pattern Recognition*, *124*, 108488.

Limited Amount of Data: Transfer Learning

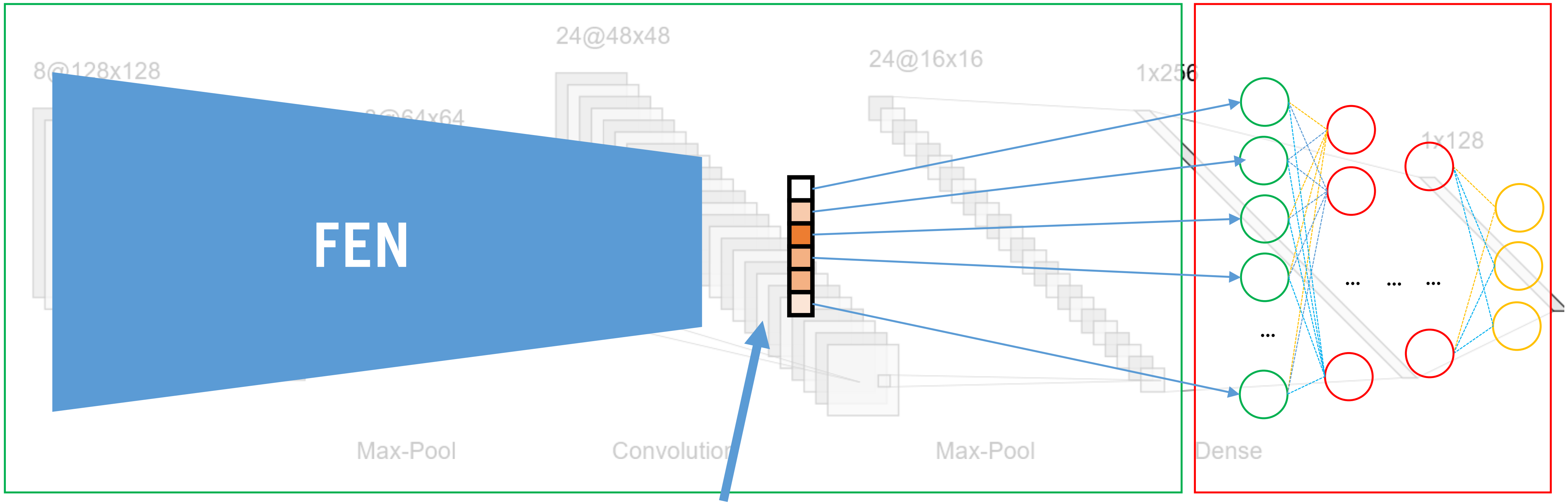
Training a CNN with Limited Aumont of Data

The Rationale Behind Transfer Learning

The typical architecture of a CNN

Convolutional and Pooling Layers
Extract high-level features from pixels (general)

Classify
(task-specific)



Latent Representation:
Data-Driven Feature Vector

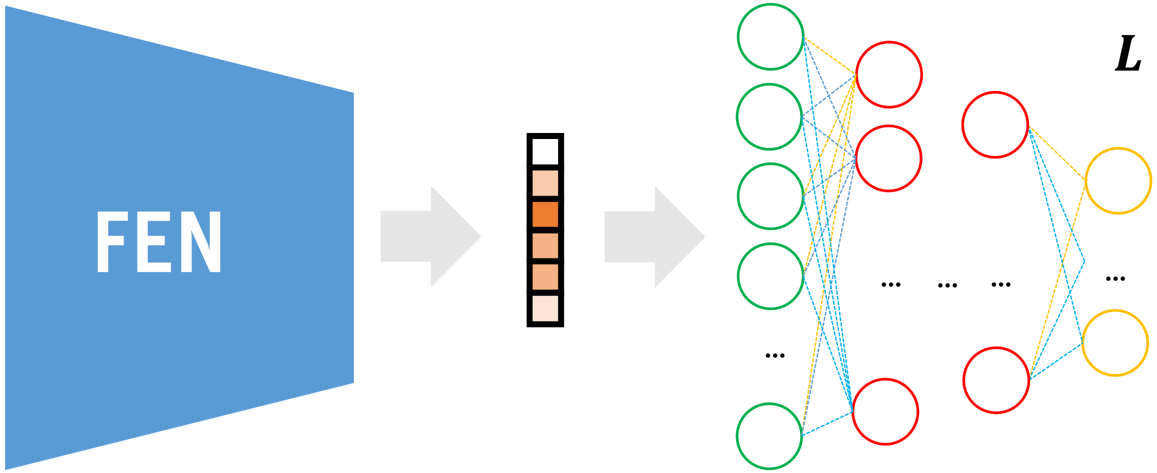
MLP for feature
classification

Very Good Features!

FEN is trained on large training sets (e.g. ImageNet) typically including hundres of classes.

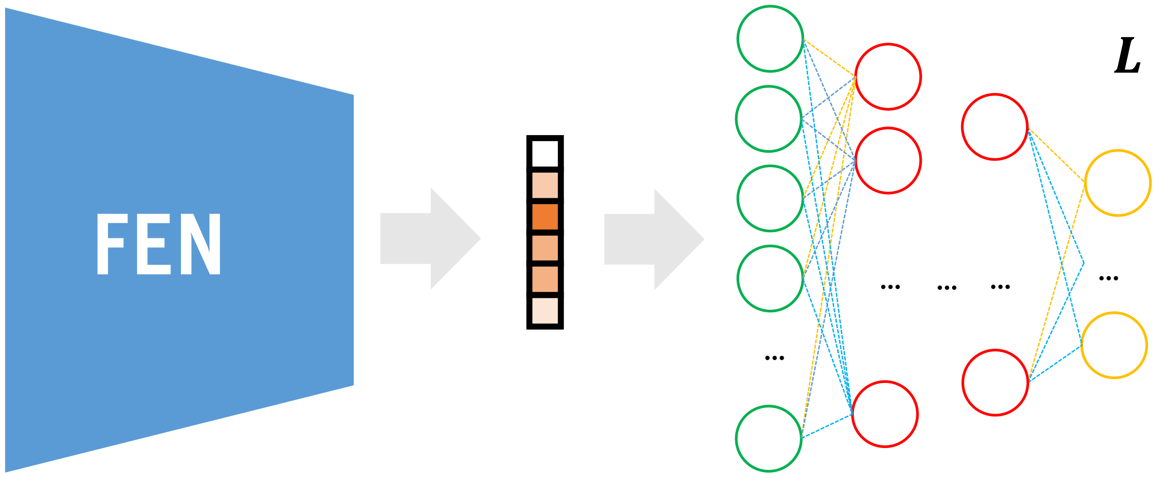
IMAGENET

$L = 999$ output neurons



0 tench, Tinca tinca
1 goldfish, Carassius auratus
2 great white shark, man-eating shark, Carcharodon caharias’,
...
998 ear, spike, capitulum
999 toilet tissue, toilet paper, bathroom tissue

Very Good Features!



$L = 999$ output neurons

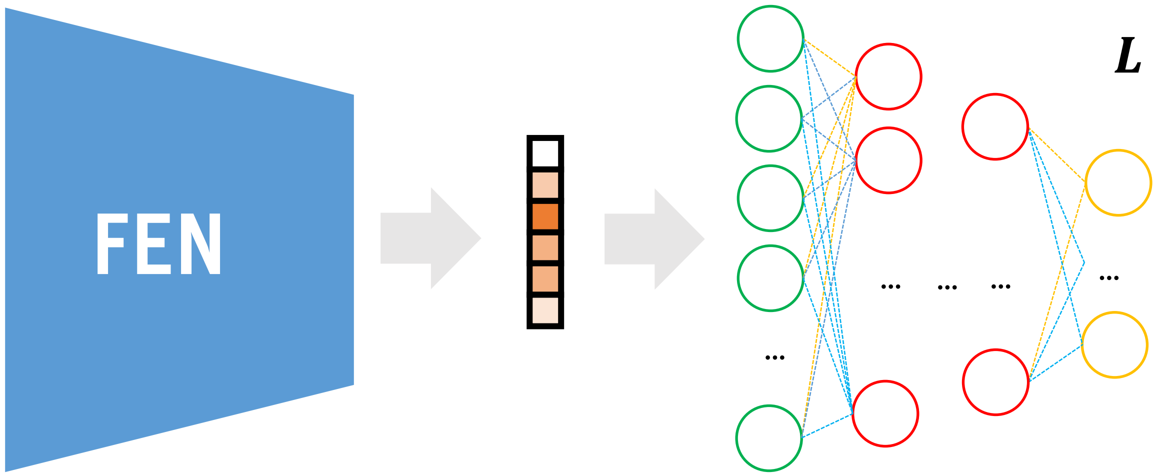
0 tench, Tinca tinca
1 goldfish, Carassius auratus
2 great white shark, man-eating shark, Carcharodon caharias',
...
998 ear, spike, capitulum
999 toilet tissue, toilet paper, bathroom tissue

The **output of the fully connected layer** has the same size as the number of classes L, and each component provide a score for the input image to belong to a specific class.

This is **very task-specific:**

- What if I have a *small* TR of images of **cats and dogs** for training?
- What if I want to train a classifier for the six types of sealions?
- Can we use these feature for solving other classification problems?

Transfer Learning

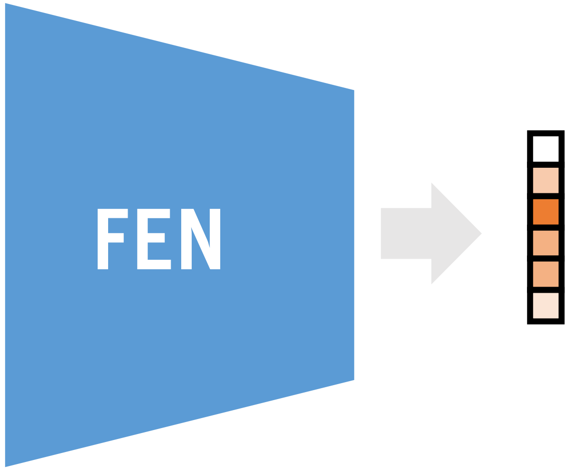


$L = 999$ output neurons

0 tench, Tinca tinca
1 goldfish, Carassius auratus
2 great white shark, man-eating shark, Carcharodon caharias',
...
998 ear, spike, capitulum
999 toilet tissue, toilet paper, bathroom tissue

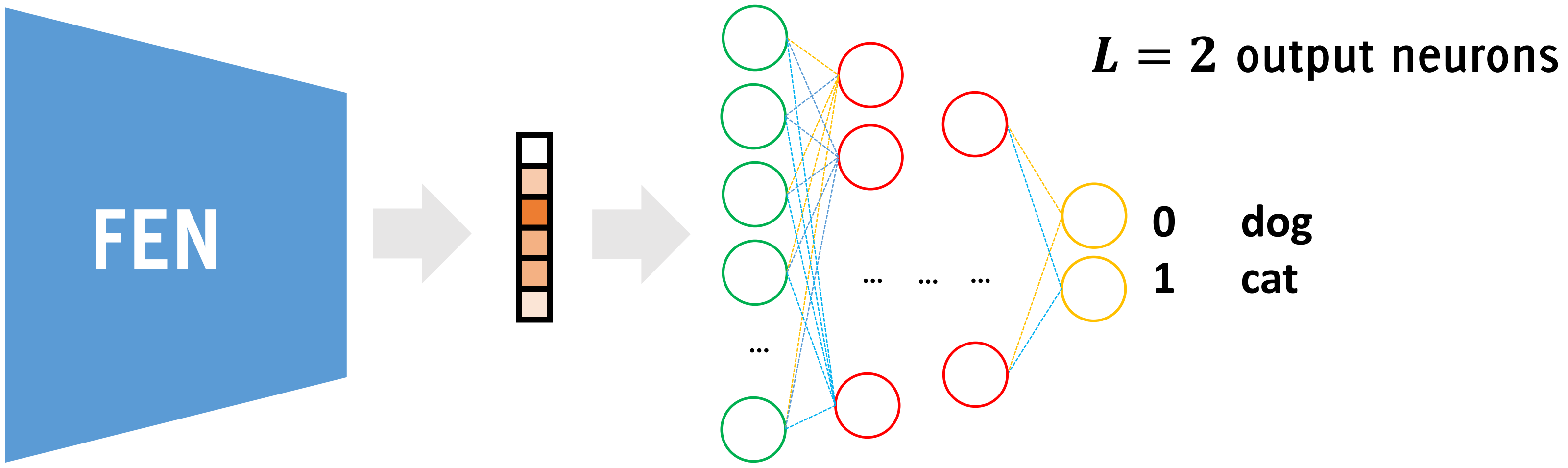
1. Take a powerful pre-trained NN (e.g., ResNet, EfficientNet, MobileNet)

Transfer Learning



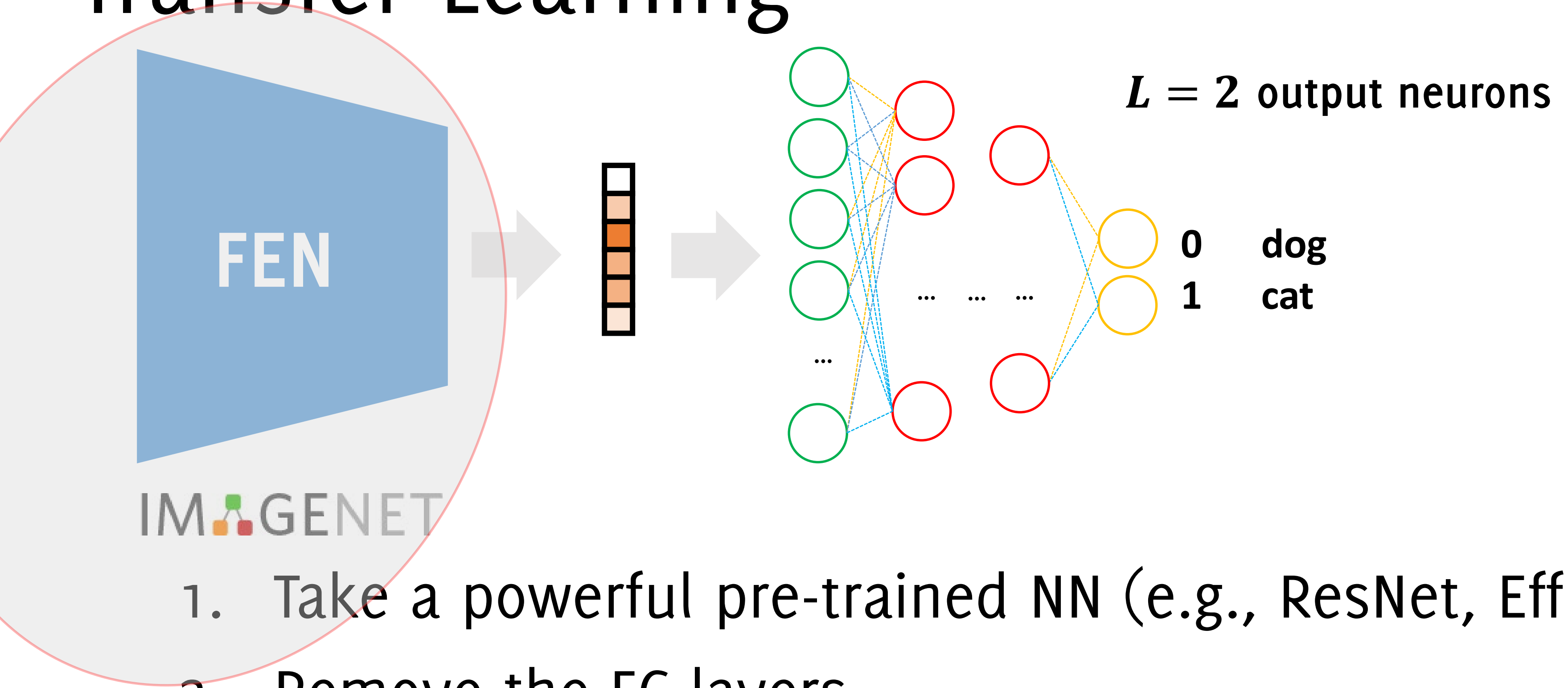
1. Take a powerful pre-trained NN (e.g., ResNet, EfficientNet, MobileNet)
2. Remove the FC layers.

Transfer Learning



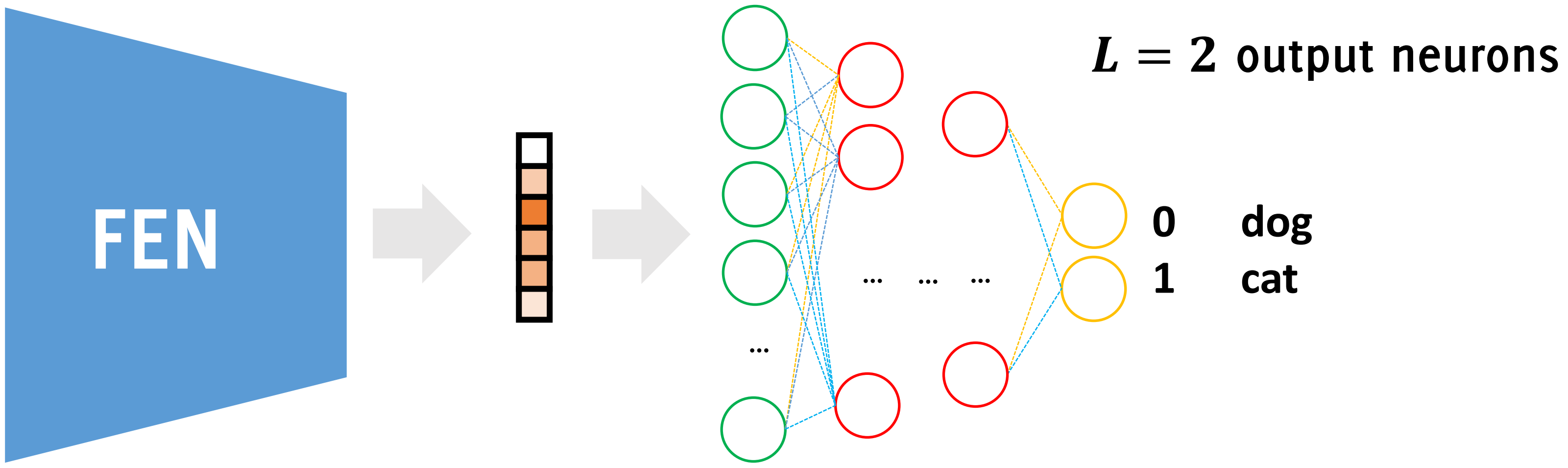
1. Take a powerful pre-trained NN (e.g., ResNet, EfficientNet, MobileNet)
2. Remove the FC layers.
3. Design new FC layers to match the new problem, plug after the FEN (initialized at random)

Transfer Learning



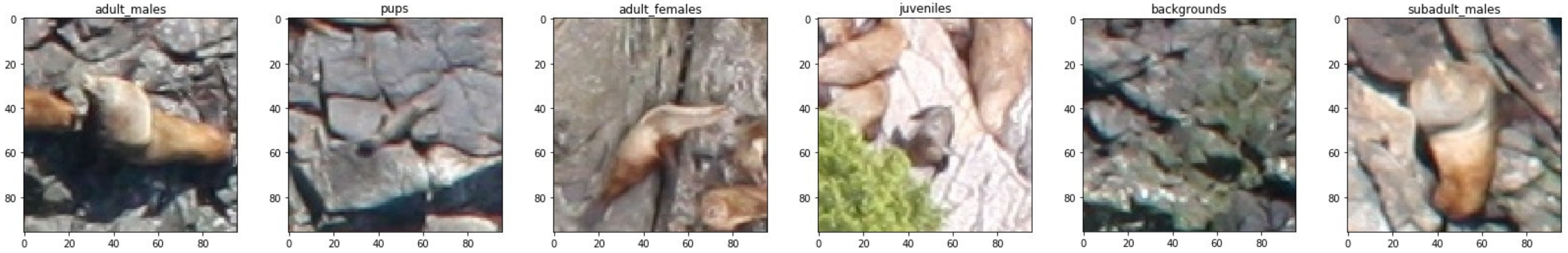
1. Take a powerful pre-trained NN (e.g., ResNet, EfficientNet, MobileNet)
2. Remove the FC layers.
3. Design new FC layers to match the new problem, plug after the FEN (initialized at random)
4. «Freeze» the weights of the FEN.

Transfer Learning

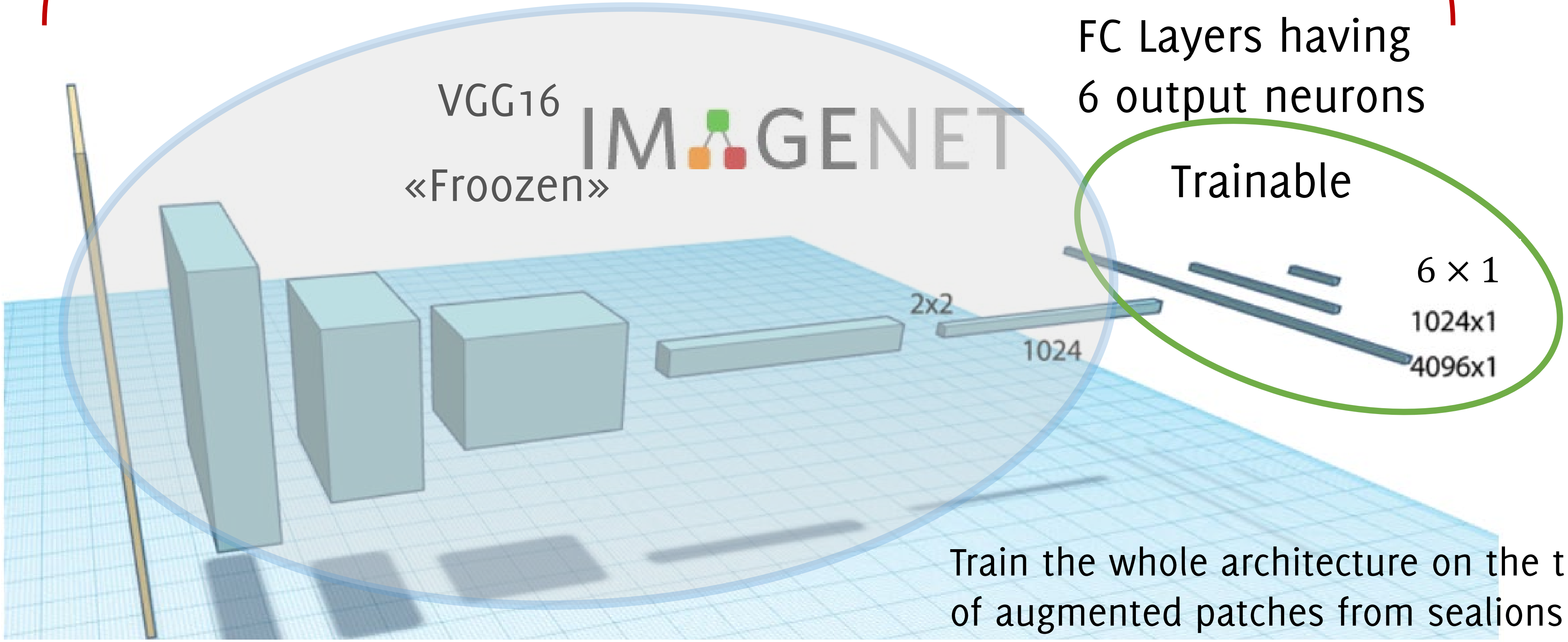


1. Take a powerful pre-trained NN (e.g., ResNet, EfficientNet, MobileNet)
2. Remove the FC layers.
3. Design new FC layers to match the new problem, plug after the FEN (initialized at random)
4. «Freeze» the weights of the FEN.
5. Train the whole network on the new training data TR

Transfer Learning in the Sealion Case



FC Layers having
6 output neurons



Train the whole architecture on the training set
of augmented patches from sealions

https://jhui.github.io/2017/03/16/CNN-Convolutional-neural-network/

Transfer Learning vs Fine Tuning

Different Options:

- **Transfer Learning: only the FC** layers are being trained. A good option when **little training data** are provided and the **pre-trained model** is expected to **match** the problem at hand
- **Fine tuning:** the whole **CNN is retrained**, but the **convolutional layers** are **initialized to the pre-trained model**. A good option when **enough training data** are provided or when the **pre-trained model is not expected to match** the problem at hand.

Typically, for the same optimizer, **lower learning rates** are used when performing fine tuning than when training from scratches

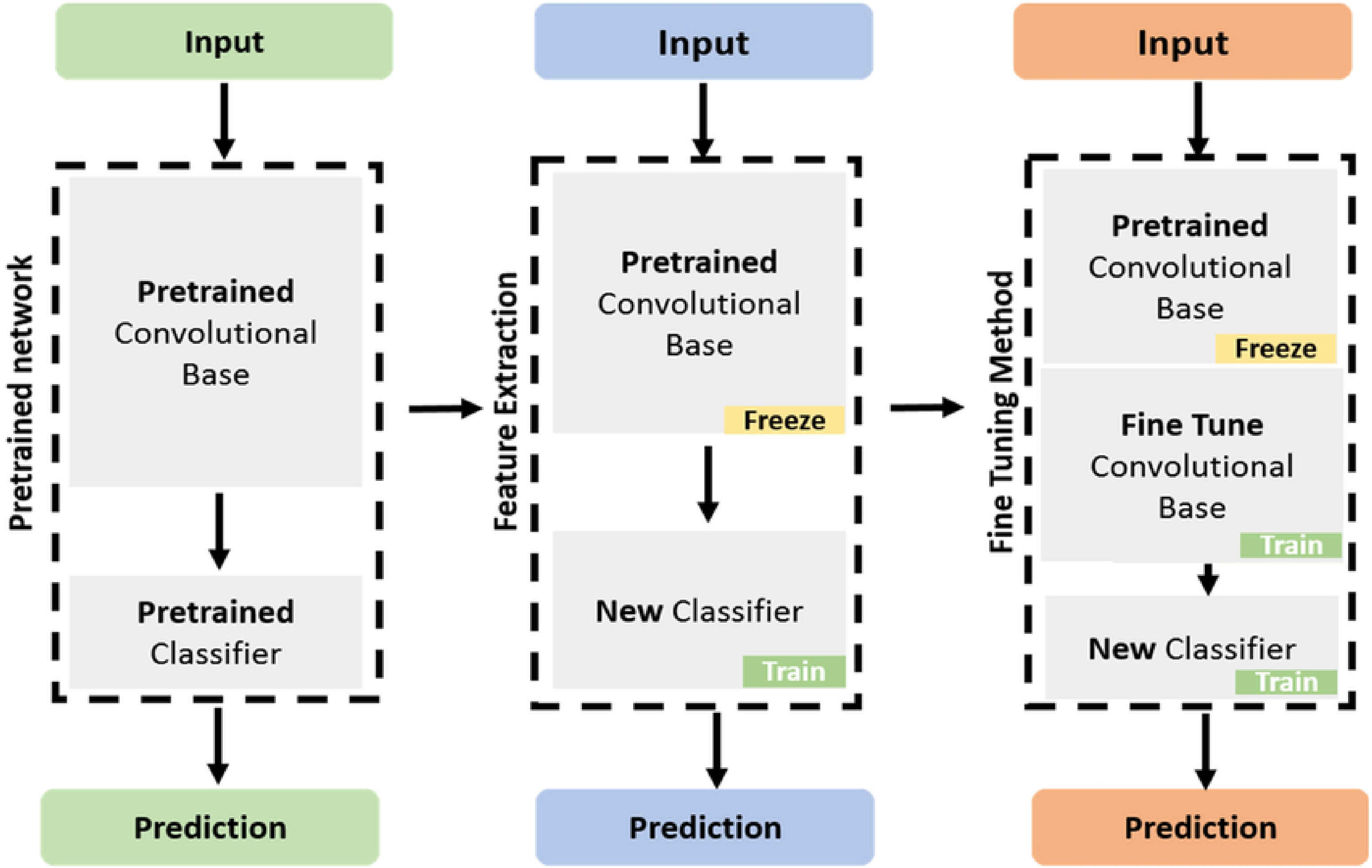
Best Practice

Typically, to take the most out of a pretrained model:

- Connect a new output layer (having few parameters)
- Transfer Learning: train the output layer only
- Make all the "last layers" trainable
- Fine tuning: train the entire network with a low learning rate

```
# Compile the model
ft_model.compile(loss=tfk.losses.BinaryCrossentropy(), optimizer=tfk.optimizers.Adam(1e-5), metrics='accuracy')
```

This strategy allows defining good predictions once the output layer has been trained


Input
Pretrained network
Pretrained Convolutional Base
Pretrained Classifier
Prediction
Input
Feature Extraction
Pretrained Convolutional Base
Freeze
New Classifier
Train
Prediction
Input
Fine Tuning Method
Pretrained Convolutional Base
Freeze
Fine Tune Convolutional Base
Train
New Classifier
Train
Prediction

Transfer Learning In Keras

Where to find pretrained models?

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

Available models

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	79.0%	94.5%	22.9M	81	109.4	8.1
VGG16	528	71.3%	90.1%	138.4M	16	69.5	4.2
VGG19	549	71.3%	90.0%	143.7M	19	84.8	4.4
ResNet50	98	74.9%	92.1%	25.6M	107	58.2	4.6
ResNet50V2	98	76.0%	93.0%	25.6M	103	45.6	4.4
ResNet101	171	76.4%	92.8%	44.7M	209	89.6	5.2
ResNet101V2	171	77.2%	93.8%	44.7M	205	72.7	5.4
ResNet152	232	76.6%	93.1%	60.4M	311	127.4	6.5
ResNet152V2	232	78.0%	94.2%	60.4M	307	107.5	6.6
InceptionV3	92	77.9%	93.7%	23.9M	189	42.2	6.9
InceptionResNetV2	215	80.3%	95.3%	55.9M	449	130.2	10.0
MobileNet	16	70.4%	89.5%	4.3M	55	22.6	3.4
MobileNetV2	14	71.3%	90.1%	3.5M	105	25.9	3.8
DenseNet121	33	75.0%	92.3%	8.1M	242	77.1	5.4
DenseNet169	57	76.2%	93.2%	14.3M	338	96.4	6.3
DenseNet201	80	77.3%	93.6%	20.2M	402	127.2	6.7
NASNetMobile	23	74.4%	91.9%	5.3M	389	27.0	6.7
NASNetLarge	343	82.5%	96.0%	88.9M	533	344.5	20.0
EfficientNetB0	29	77.1%	93.3%	5.3M	132	46.0	4.9

Importing Pretrained Models in keras…

Pre-trained models are available, typically in two ways:

- **`include_top = True`:** provides the entire network, including the fully convolutional layers. This network can be used to solve the classification problem it was trained for
- **`include_top = False`:** contains only the convolutional layers of the network, and it is specifically meant for transfer learning.

Have a look at the size of these models in the two options!

Importing Pretrained Models in keras…

```
from keras import applications
base_model = applications.VGG16(weights =
"imagenet", include_top=False, input_shape =
(img_width, img_width, 3), pooling = “avg")
```

Importing Pretrained Models in keras…

```
from keras import applications
base_model = applications.VGG16(weights =
"imagenet", include_top=False, input_shape =
(img_width, img_width, 3), pooling = “avg")
```

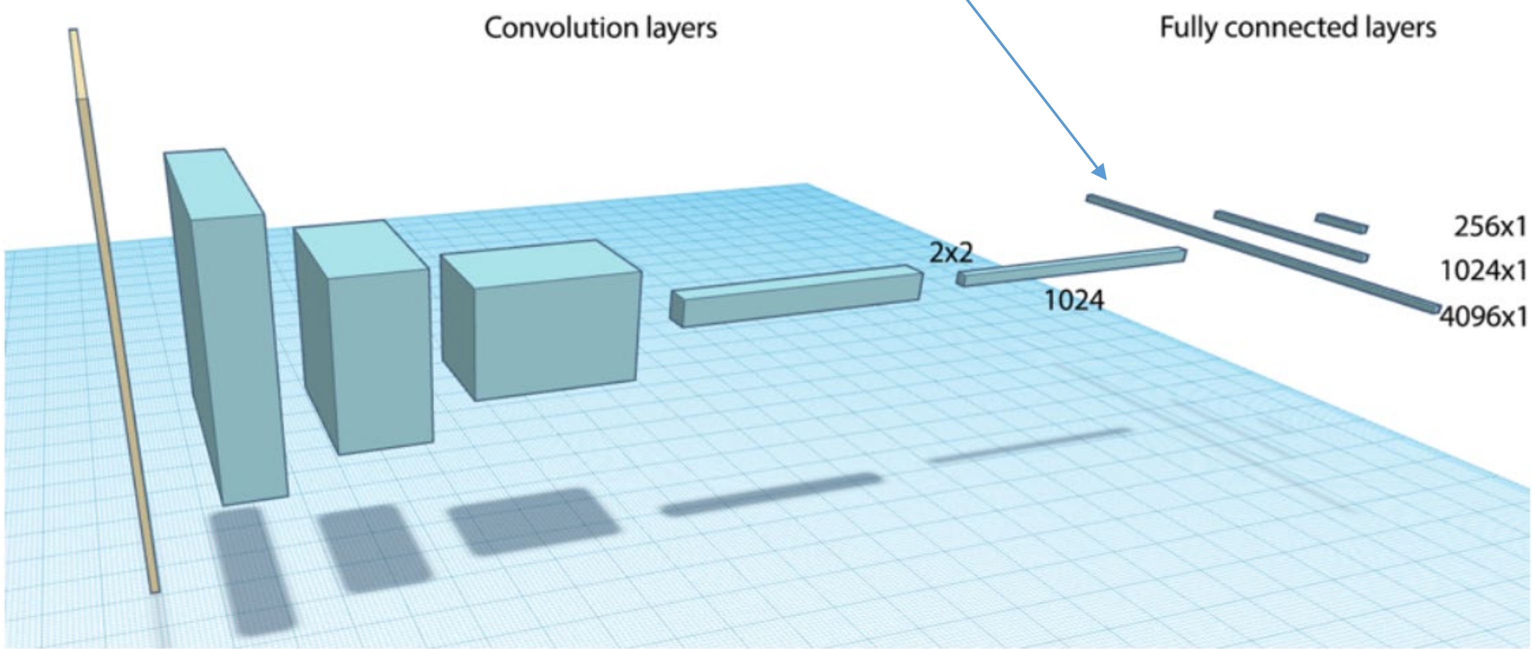
When **`include_top=False`**, the network returns the output of a global pooling layer, which can be:

- **`pooling = “avg“`** Global Averaging Pooling (GAP)
- **`pooling = “max“`** Global Max Pooling (GMP)
- **`pooling = “none“`** There is no pooling, it returns the activations

How to extract the feature extraction network?

Actually, for sequential models, you create feature extraction network

```
fen = tfk.Sequential(model.layers[:-2])
fen.output_shape
>> 128
```



How to extract the feature extraction network?

Actually, for sequential models, you create feature extraction network

```
fen = tfk.Sequential(model.layers[:-2])
```

Note: each Keras Application expects a specific kind of input **preprocessing**.

For MobileNetV2, call

```
tf.keras.applications.mobilenet_v2.preprocess_input
```

on your inputs before passing them to the model. mobilenet_v2.preprocess_input will scale input pixels between -1 and 1.

Transfer Learning in Keras…

Requires a bit of TensorFlow Backend to add the modified Fully connected layer at the top of a pretrained model

Then, before training it is necessary to loop through the network layers (they are in **`model.layers`**) and then modify the trainable property

```
for layer in model.layers[: lastFrozen]:
    layer.trainable=False
```

An example of model loading

```
# load a pre-
trained MobileNetV2 model without weights
mobile = tfk.applications.MobileNetV2(
    input_shape=(224, 224, 3),
    include_top=False,
    pooling='avg',
)
```

Transfer Learning: adding the new Network Top

Requires a bit of TensorFlow Backend to add the modified Fully connected layer at the top of a pretrained model

Then, before training it is necessary to loop through the network layers (they are in **`model.layers`**) and then modify the trainable property

```
# Add the classifier layer to the MobileNet
inputs = tfk.Input(shape=(224,224,3))
x = mobile(inputs) # concatenates inputs and the output of the pretrined network... the entire mobileNet is handled as a layer
x = tfkl.Dropout(0.5)(x) # good to prevent overfitting
outputs = tfkl.Dense(1, activation='sigmoid')(x) # connect a new output layer
```

Transfer Learning: setting layers trainable property

Requires a bit of TensorFlow Backend to add the modified Fully connected layer at the top of a pretrained model

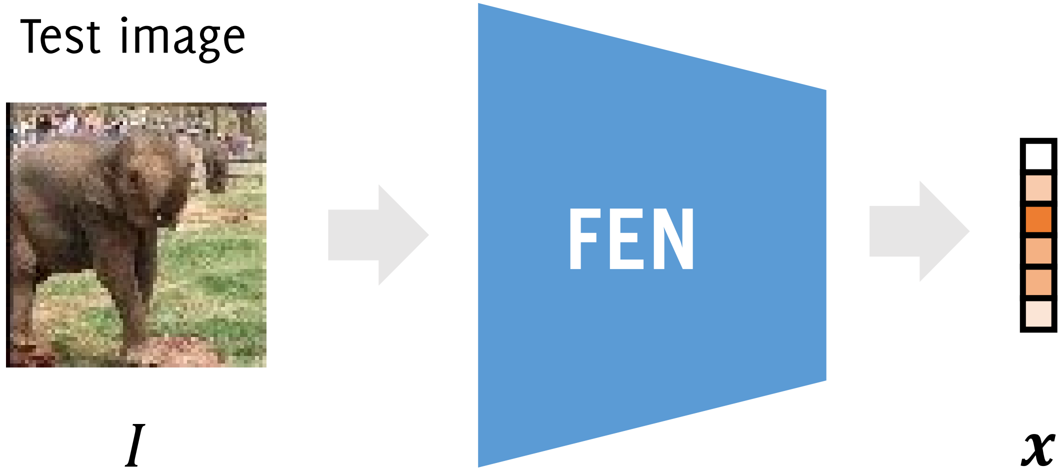
Then, before training it is necessary to loop through the network layers (they are in **`model.layers`**) and then modify the trainable property

```
for layer in model.layers[: lastFrozen]:
    layer.trainable=False
```

Image Retrieval From The Latent Space

Features are Good For Image Retrieval

Feed a test image and compute its latent representation



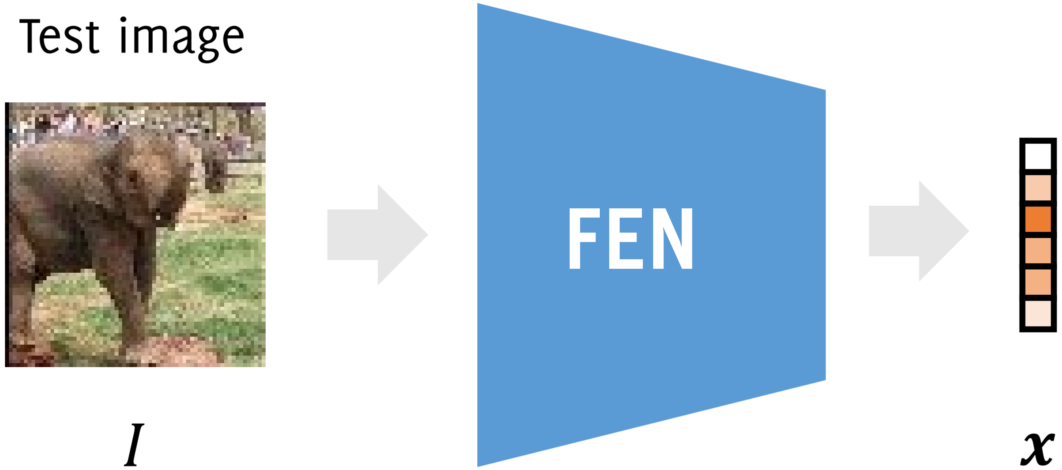
Latent Representation:
Data-Driven Feature Vector

Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton. "Imagenet classification with deep convolutional neural networks." NIPS 2012

Features are Good For Image Retrieval

Feed a test image and compute its latent representation

Test image

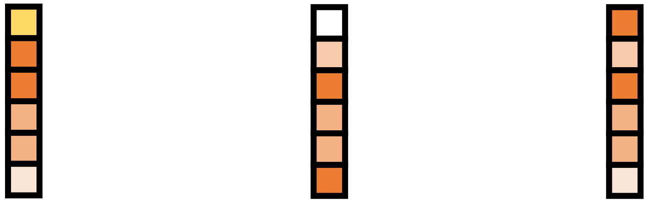


I $\qquad$ $\boldsymbol{x}$

Latent Representation:
Data-Driven Feature Vector

Retrieve the training images having the closest latent representations

The 3- nearest neighborhood of $\boldsymbol{x}$



$\boldsymbol{x}_1$ $\qquad$ $\boldsymbol{x}_2$ $\qquad$ $\boldsymbol{x}_3$

Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton. "Imagenet classification with deep convolutional neural networks." NIPS 2012

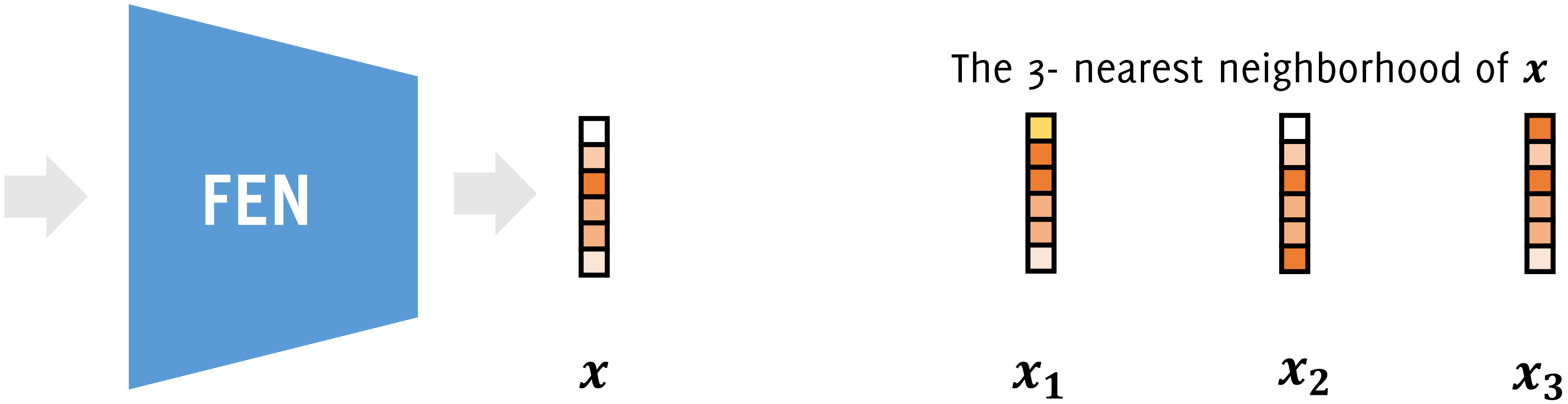
Features are Good For Image Retrieval

Feed a test image and compute its latent representation

Retrieve the training images having the closest latent representations

Test image

I



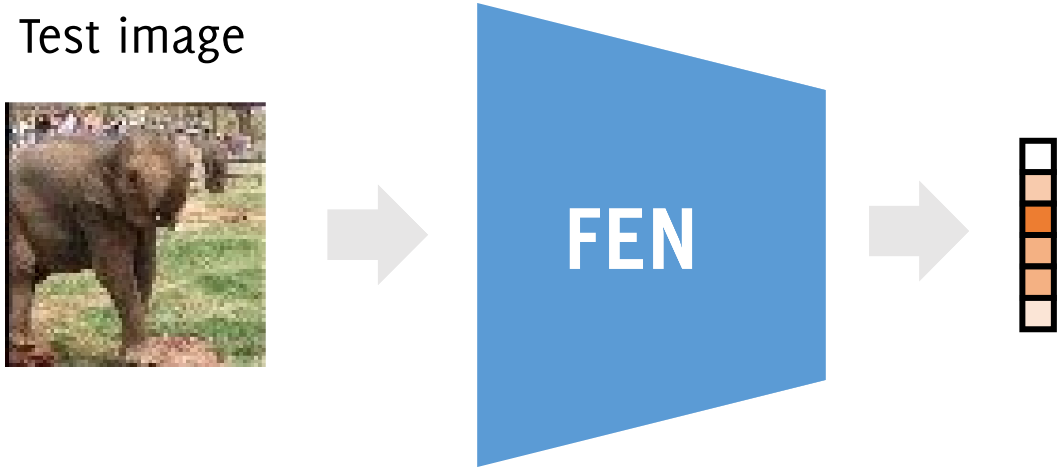
Latent Representation:
Data-Driven Feature Vector

Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton. "Imagenet classification with deep convolutional neural networks." NIPS 2012

Features are Good For Image Retrieval

Feed a test image and compute its latent representation

Test image



Training Images corresponding to the closest latent representations!

Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton. "Imagenet classification with deep convolutional neural networks." Advances in neural information processing systems 25 (2012).

1-NN classification in the latent space

```
# feed the test imate to the fen
image_features = fen.predict(test_image)


# feed fen with the entire training set (use batches of 512)
features = fen.predict(X_train_val,batch_size=512,verbose=0)


# compute distances (e.g. ell1) between image_featres and features,
distances = np.mean(np.abs(features - image_features),axis=-1)
sortedDistances = distances.argsort()


# sort images (and labels) according to the distance computed above
ordered_images = X_train_val[sortedDistances]
ordered_labels = y_train_val[sortedDistances]
# associate to image_features the closest image ordered_images[0]
```