Course: Deep Learning

Unit 2: Computer Vision

Convolutional Neural Networks (II): Regularization

Luis Baumela

Universidad Politécnica de Madrid

Convolutional Neural Networks (II)

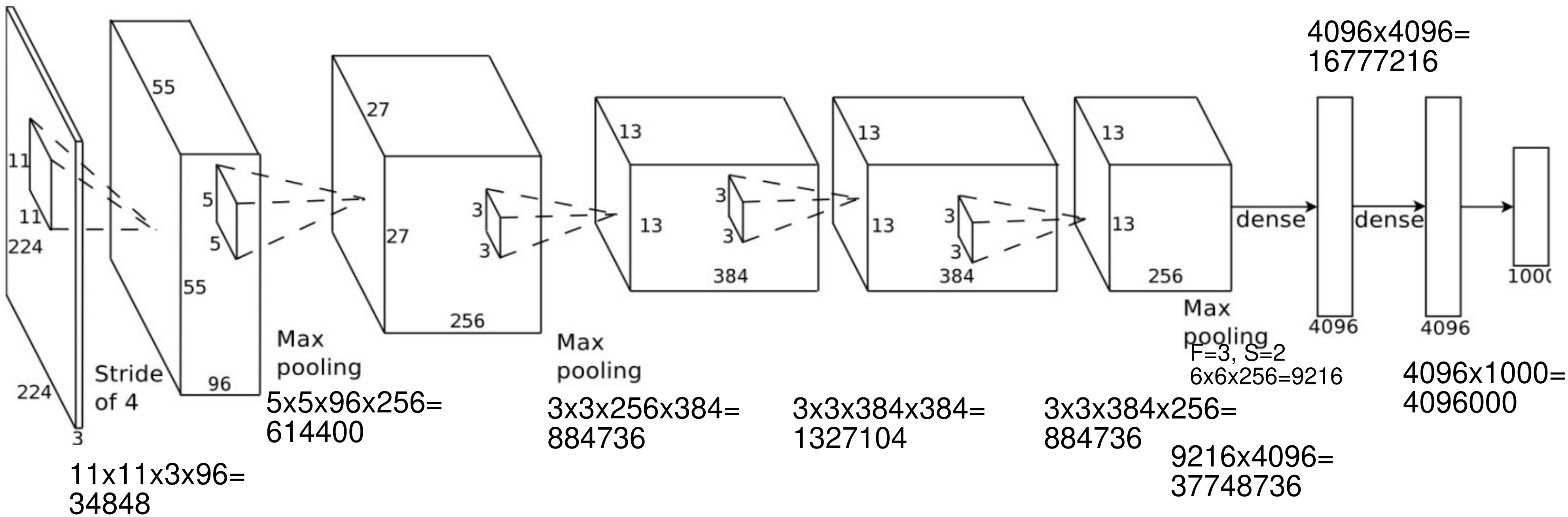
1. Introduction
    - Why regularize?

2. Regularization techniques
    - Data augmentation
    - Dropout
    - Transfer learning
    - Label Smoothing

Introduction

- Why regularize?

  How many parameters in Alexnet CNN?



Conv Layers: 3.7 M params

ffNN Layers: 58.6 M params

Total: 62.3 M params

ILSVRC: 1.2 M training images

94% of params

Introduction

- Why regularize?

How many parameters in Alexnet CNN? **62.3 M**
How many training images? **1.2 M**

What will happen if we do not care at all?
Overfit the training data set → **poor generalization**

Solution:
Reduce the degrees of freedom in your model → **Regularize**

Regularization

- Data augmentation

  Scarce data is the main bottleneck in deep models.

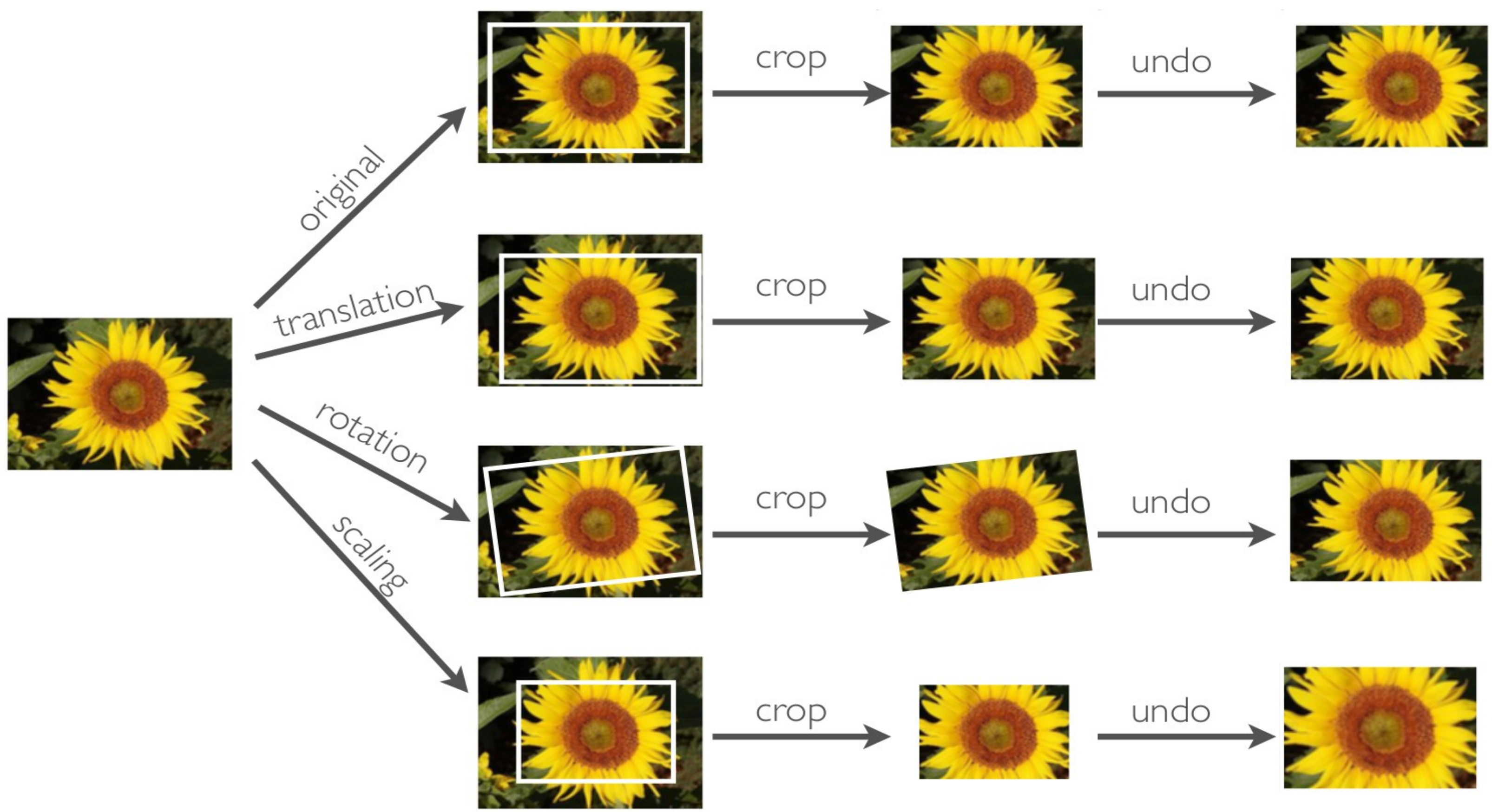
  - CNNs have some built-in invariances:
    small translations due to convolution and max pooling

  - Not invariant to other important variations such as rotations, scale, colour changes, noise, etc..

  - However, it's easy to artificially generate data with such transformations
    ➔ use such data as additional training data
    ➔ NN will learn to be invariant to such transformations

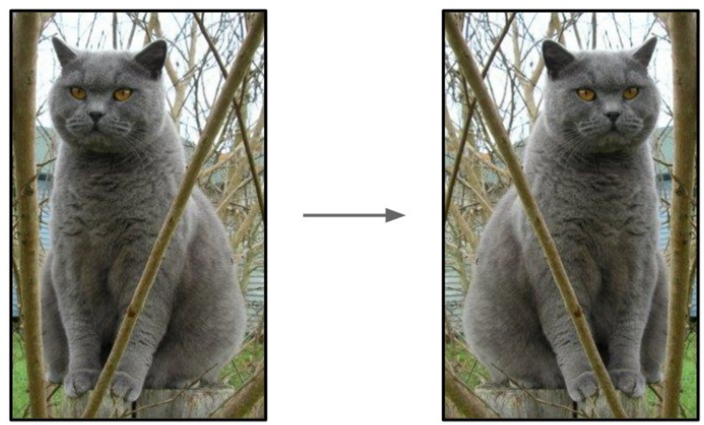
Regularization

- Data augmentation (different pixels, same label)
  - Translation, rotation, scale, shear changes:



Regularization

- Data augmentation (different pixels, same label)
  - Translation, rotation, scale, shear changes
  - Horizontal flips



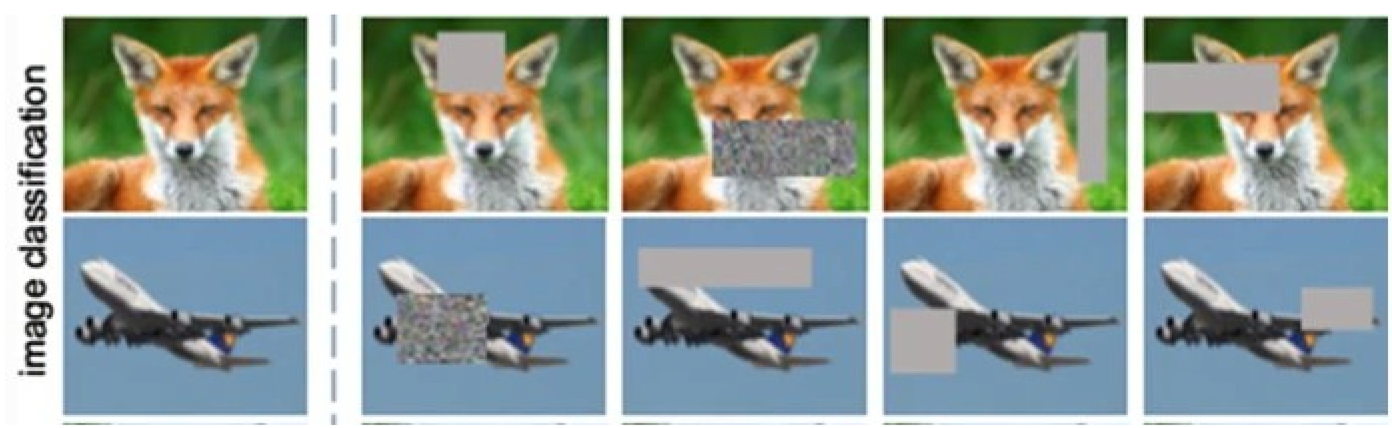
Regularization

- Data augmentation (different pixels, same label)
  - Translation, rotation, scale, shear changes
  - Horizontal flips
  - Pixel value manipulation

Shorten. A survey on Image Data Augmentation for Deep Learning. Journal of Big Data, 2019

Regularization

- Data augmentation (different pixels, same label)
  - Translation, rotation, scale, shear changes
  - Horizontal flips
  - Pixel value manipulation
  - Random pixel erasing (**CutOut**)



Shorten. A survey on Image Data Augmentation for Deep Learning. Journal of Big Data, 2019

Regularization

- Data augmentation (different pixels, same label)
  - Translation, rotation, scale, shear changes
  - Horizontal flips
  - Pixel value manipulation
  - Random pixel erasing
  - Synthetic data generation

Regularization

- Data Augmentation.
  - **MixUp**

Obtaining new training data through convex combinations of pairs of examples and their labels.

Builds synthetic training examples by

$\tilde{x} = \lambda x_i + (1 - \lambda) x_j,$ where x_i, x_j are raw input vectors

$\tilde{y} = \lambda y_i + (1 - \lambda) y_j,$ where y_i, y_j are one-hot label encodings

$\tilde{x} =$ 

$$\tilde{y} = 0.3 * y_{cat} + 0.7 * y_{dog}$$

Zhang et al. Mixup: Beyond empirical risk minimization. arXiv preprint arXiv:1710.09412, 2017.

Regularization

- Data Augmentation. **CutMix**
  Overlays random regions in training images with a patch of from another.

$$\tilde{x} = \mathbf{M} \odot x_A + (\mathbf{1} - \mathbf{M}) \odot x_B$$
$$\tilde{y} = \lambda y_A + (1 - \lambda) y_B,$$

where (x_A, y_A) and (x_B, y_B) training samples
$(\tilde{x}, \tilde{y})$ generated data
$\mathbf{M} \in \{0,1\}^{W \times H}$ binary matrix
λ combination ratio

	CutMix
Image	
Label	Dog 0.6 Cat 0.4

- The rectangular masc is given by

$$r_x \sim \text{Unif}\ (0, W), \quad r_w = W\sqrt{1-\lambda},$$
$$r_y \sim \text{Unif}\ (0, H), \quad r_h = H\sqrt{1-\lambda}$$
$$\frac{r_w r_h}{WH} = 1 - \lambda$$

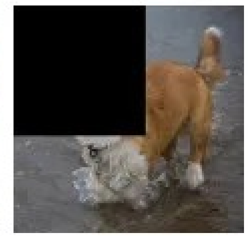
where

W: image width
H: image heigth
r_x: x cut co-ordinate
r_y: y cut co-ordinate
r_w: cut width
r_h: cut heigth

Zhang et al. Mixup: Beyond empirical risk minimization. arXiv preprint arXiv:1710.09412, 2017.

Regularization

- Data Augmentation. CutMix vs. MixUp vs. CutOut

	ResNet-50	Mixup	Cutout	CutMix
Image				
Label	Dog 1.0	Dog 0.5 Cat 0.5	Dog 1.0	Dog 0.6 Cat 0.4
ImageNet Cls (%)	76.3 (+0.0)	77.4 (+1.1)	77.1 (+0.8)	**78.4** **(+2.1)**
ImageNet Loc (%)	46.3 (+0.0)	45.8 (-0.5)	46.7 (+0.4)	**47.3** **(+1.0)**
Pascal VOC Det (mAP)	75.6 (+0.0)	73.9 (-1.7)	75.1 (-0.5)	**76.7** **(+1.1)**

Zhang et al. Mixup: Beyond empirical risk minimization. arXiv preprint arXiv:1710.09412, 2017.

Regularization

- Data augmentation (different pixels, same label)
  - Translation, rotation, scale, shear changes
  - Horizontal flips
  - Pixel value manipulation
  - Random pixel erasing
  - Synthetic data generation

- At testing

Average network response at a fixed set of tranformations

Regularization

- Data augmentation (different pixels, same label)
  Some support from Keras

```
keras.preprocessing.image.ImageDataGenerator(featurewise_center=False, samplewise_center
```

- **featurewise_center**: Boolean. Set input mean to 0 over the dataset, feature-wise.
- **samplewise_center**: Boolean. Set each sample mean to 0.
- **featurewise_std_normalization**: Boolean. Divide inputs by std of the dataset, feature-wise.
- **samplewise_std_normalization**: Boolean. Divide each input by its std.
- **zca_epsilon**: epsilon for ZCA whitening. Default is 1e-6.
- **zca_whitening**: Boolean. Apply ZCA whitening.
- **rotation_range**: Int. Degree range for random rotations.
- **width_shift_range**: Float, 1-D array-like or int

- **horizontal_flip**: Boolean. Randomly flip inputs horizontally.
- **vertical_flip**: Boolean. Randomly flip inputs vertically.
- **rescale**: rescaling factor. Defaults to None. If None or 0, no rescaling is applied, otherwise we multiply the data by the value provided (after applying all other transformations).
- **preprocessing_function**: function that will be implied on each input. The function will run after the image is resized and augmented. The function should take one argument: one image (Numpy tensor with rank 3), and should output a Numpy tensor with the same shape.

Regularization

- Data augmentation (different pixels, same label)
  Some support from Keras

```
keras.preprocessing.image.ImageDataGenerator(featurewise_center=False, samplewise_center
```

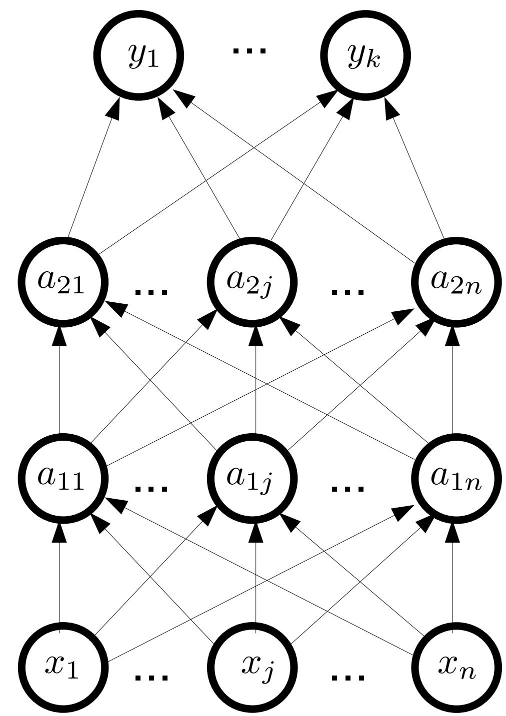
```
(x_train, y_train), (x_test, y_test) = cifar10.load_data()
y_train = np_utils.to_categorical(y_train, num_classes)
y_test = np_utils.to_categorical(y_test, num_classes)

datagen = ImageDataGenerator(
    featurewise_center=True,
    featurewise_std_normalization=True,
    rotation_range=20,
    width_shift_range=0.2,
    height_shift_range=0.2,
    horizontal_flip=True)

# compute quantities required for featurewise normalization
# (std, mean, and principal components if ZCA whitening is applied)
datagen.fit(x_train)

# fits the model on batches with real-time data augmentation:
model.fit_generator(datagen.flow(x_train, y_train, batch_size=32),
                    steps_per_epoch=len(x_train) / 32, epochs=epochs)
```

Regularization

- Dropout

  Prune the NN by ramdomly discarding hidden units

  The output of each hidden unit is multiplied by “0” with a probability p (for example p=0.5)

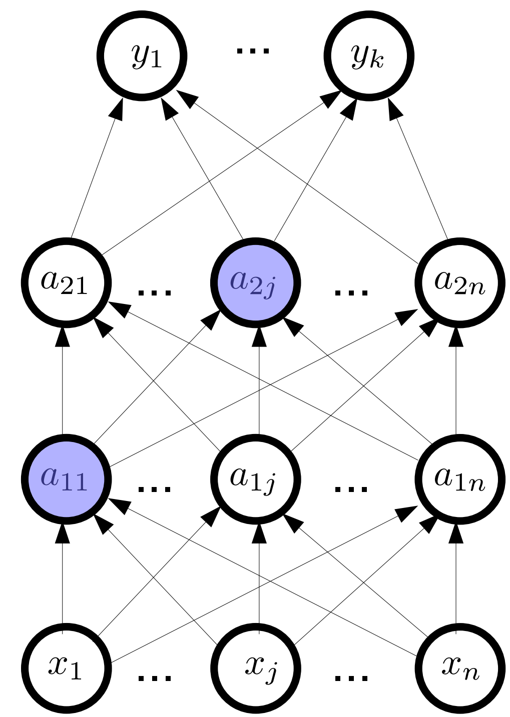


Regularization

- Dropout

  Prune the NN by ramdomly discarding hidden units

  The output of each hidden unit is multiplied by “0” with a probability p (for example p=0.5)



Regularization

- Dropout

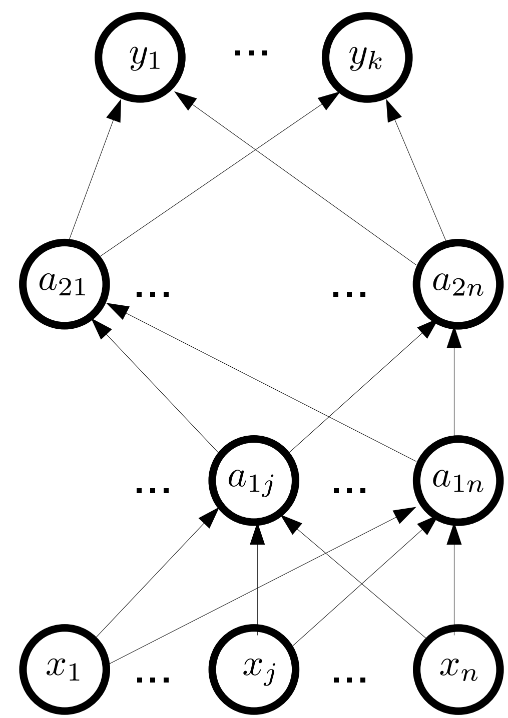
  Prune the NN by ramdomly discarding hidden units

  The output of each hidden unit is multiplied by “0” with a probability p (for example p=0.5). Now

  - Hidden units cannot co-adapt
  - Hidden units must learn more general features

  Changes on the **training algorithm**:

  - Forwprop. Each layer includes a random [0,1] mask that multiplies each activation.
  - Backprop. Hidden units multiplied by “0” do not contribute to the gradient backprop.



Regularization

- Dropout, in Keras

```
keras.layers.Dropout(rate, noise_shape=None, seed=None)
```

Applies Dropout to the input.

Dropout consists in randomly setting a fraction `rate` of input units to 0 at each update during training time, which helps prevent overfitting.

Arguments

- **rate**: float between 0 and 1. Fraction of the input units to drop.
- **noise_shape**: 1D integer tensor representing the shape of the binary dropout mask that will be multiplied with the input. For instance, if your inputs have shape `(batch_size, timesteps, features)` and you want the dropout mask to be the same for all timesteps, you can use `noise_shape=(batch_size, 1, features)`.
- **seed**: A Python integer to use as random seed.

Regularization

- Transfer learning (pre-training)

  What to do if we have a problem with a small data set.

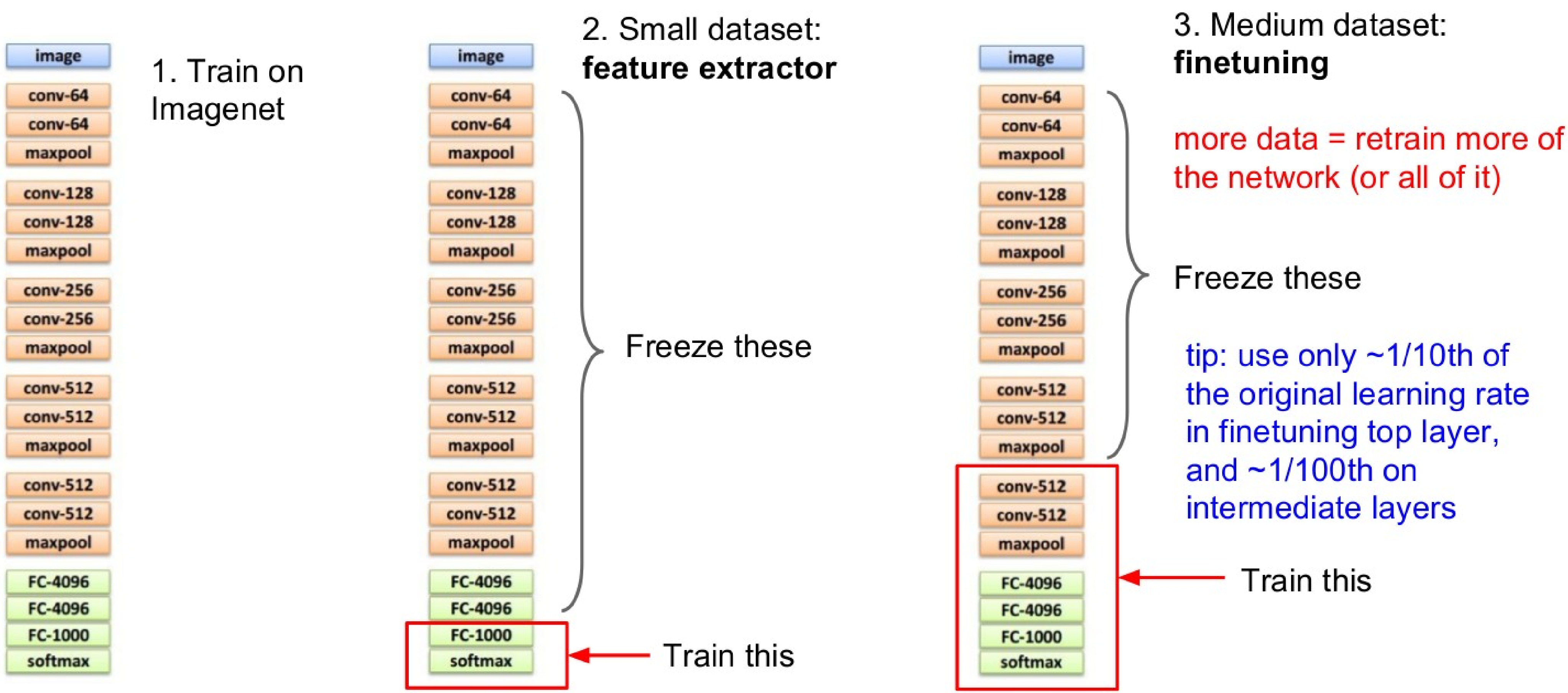
  1. Use another (similar) large data set to train a large CNN.
  2. Re-train the large CNN with your small data set.

  Take advantage of

  - Keras applications package
  - pyTorch, Tensor Flow, Caffe model zoo models in the net
  - Follow procedures to convert from one model to another

Regularization

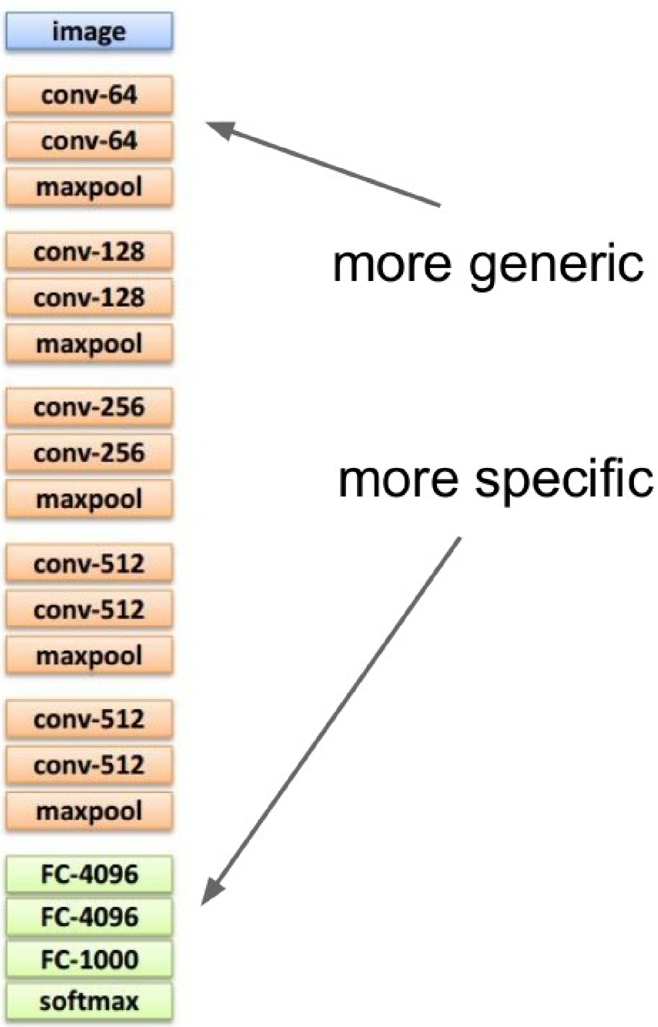
- Transfer learning (pre-training)



http://cs231n.stanford.edu/

Regularization

- Transfer learning (pre-training)

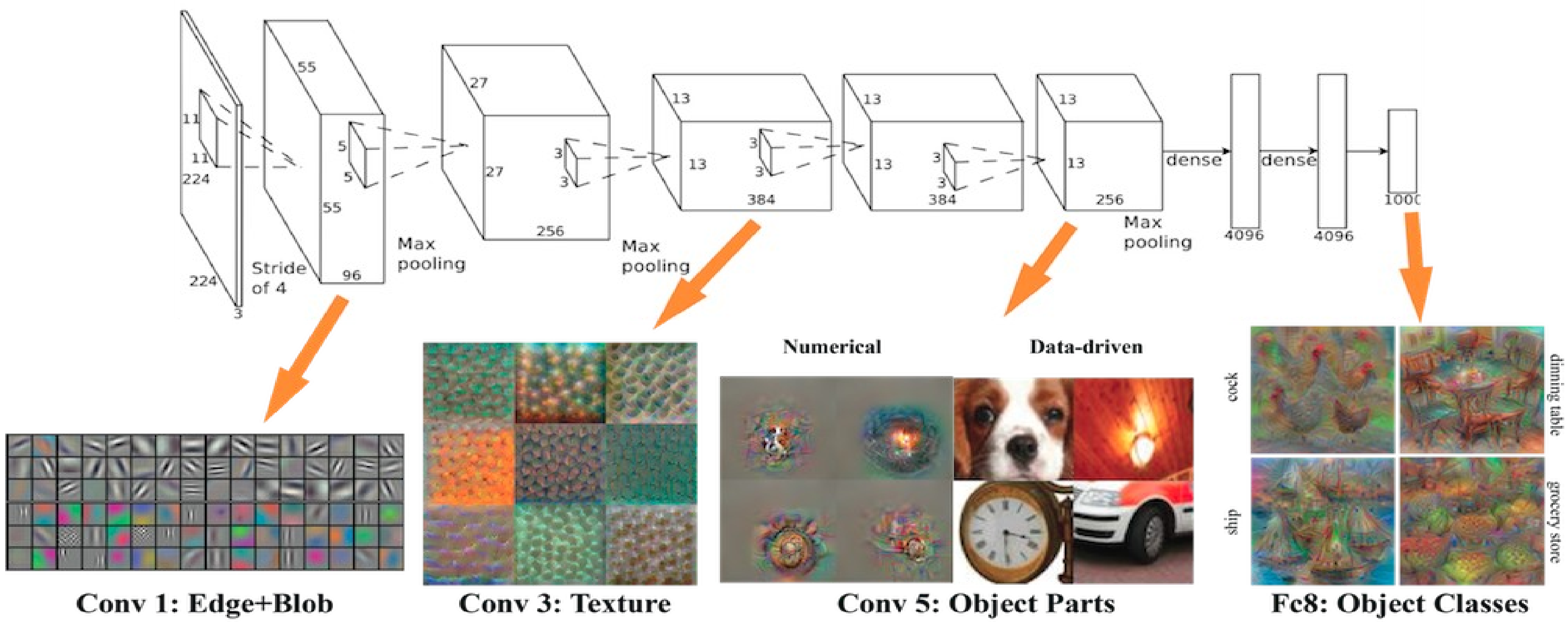


	very similar dataset	**very different dataset**
very little data	Use Linear Classifier on top layer	You’re in trouble… Try linear classifier from different stages
quite a lot of data	Finetune a few layers	Finetune a larger number of layers

http://cs231n.stanford.edu/

Regularization

- Transfer learning as representation learning



Regularization

- Transfer learning in Keras

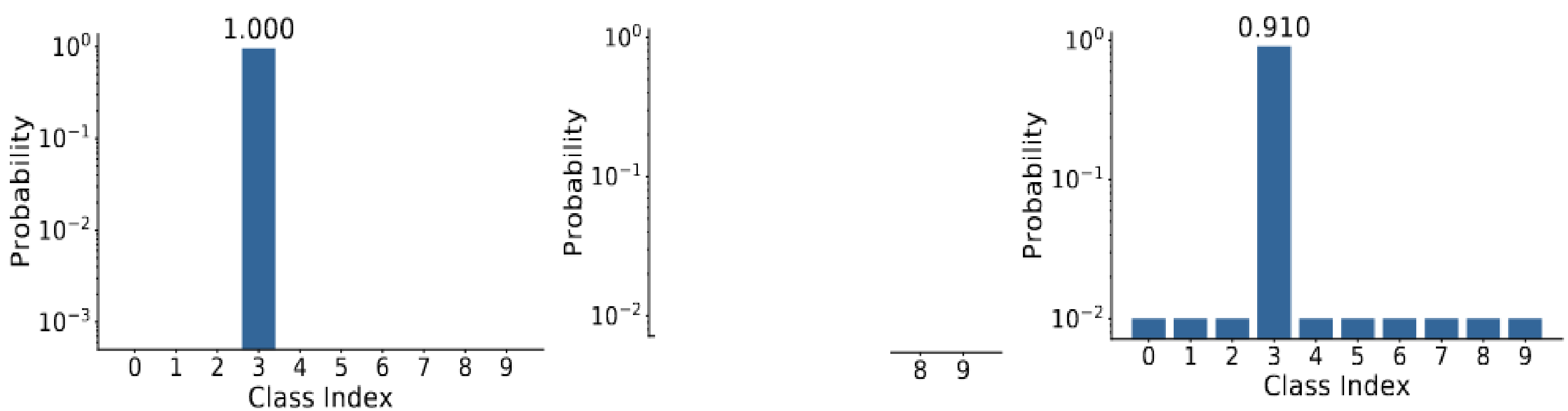
```
from keras.applications.inception_v3 import InceptionV3
from keras.preprocessing import image
from keras.models import Model
from keras.layers import Dense, GlobalAveragePooling2D
from keras import backend as K

# create the base pre-trained model
base_model = InceptionV3(weights='imagenet', include_top=False)

# add a global spatial average pooling layer
x = base_model.output
x = GlobalAveragePooling2D()(x)
# let's add a fully-connected layer
x = Dense(1024, activation='relu')(x)
# and a logistic layer -- let's say we have 200 classes
predictions = Dense(200, activation='softmax')(x)

# this is the model we will train
model = Model(inputs=base_model.input, outputs=predictions)

# first: train only the top layers (which were randomly initialized)
# i.e. freeze all convolutional InceptionV3 layers
for layer in base_model.layers:
    layer.trainable = False

# compile the model (should be done *after* setting layers to non-trainable)
model.compile(optimizer='rmsprop', loss='categorical_crossentropy')

# train the model on the new data for a few epochs
model.fit_generator(...)
```

Regularization

- Label smoothing

  A model is *overconfident* when its confidence on its predictions is higher that the actual accuracy.

  Label Smoothing is a regularization technique that introduces noise in the labels.



(a) Hard Label

(b) LS

$$y_{hot} = [0, 0, 1, \ldots, 0] \qquad y_{soft} = (1 - \alpha) * y_{hot} + \frac{\alpha}{K}$$

Regularization

- Label smoothing

  A model is *overconfident* when its confidence on its predictions is higher that the actual accuracy.
  Label Smoothing is a regularization technique that introduces noise in the labels.

  Benefits:
  - Prevents overfitting.
  - Improves generalization.
  - Stabilizes training.
  - Enhanced Calibration.

  Drawbacks:
  - One extra hyperparameter to tune
  - May not be beneficial highly imbalanced data sets.

Regularization

- Label smoothing in Keras

tf.keras.losses.CategoricalCrossentropy

Computes the crossentropy loss between the labels and predictions.

View aliases

```
tf.keras.losses.CategoricalCrossentropy(
    from_logits=False, label_smoothing=0, reduction=losses_utils.ReductionV2.AUTO,
    name='categorical_crossentropy'
)
```